UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO DE FIN DE GRADO

MÉTODOS DE APRENDIZAJE AUTOMÁTICO PARA EL PROCESAMIENTO DE ARRAYS DE GENES

AUTOR: ICÍAR CIVANTOS GÓMEZ

TUTOR: VANESSA GÓMEZ VERDEJO

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

ÍNDICE DE EXPRESIONES

INTRODUCTION

The main purpose of this research is to carry out the processing and the classification of several arrays of genes in order to resolve which of them become relevant for the diagnosis of certain diseases. The DNA sequencing techniques have allowed us to obtain genetic chains composed by thousands of units. In spite of all of these genes play an important role in our organism, we do not need the entire sequence for detecting a concrete illness. Each part of the array provides some information about a certain aspect of our body. Due to each one affects directly or indirectly to the pathology in question, the objective of this investigation is to elaborate a study in which, by using a machine learning algorithm, we come to be able to conclude what elements are more decisive when we try to classify some chains. In this way, we become capable to detect quickly if a concrete illness is going to appear in a person or not.

With this aim we have decided to implement a linear classifier. Even though a lot of them could be use for this purpose, because of computational cost reasons and its effectiveness we have decided to employ the support vector machines algorithm (SVM). It will receive databases of genetic sequences which belong to both healthy and ill patients. These sets have been previously grouped in function of the illnesses under study so that we can compare then what are the differences between these chains. Using these results we will elaborate a ranking of the genes we consider most relevant when we are looking for detecting pathology.

In this process we will make use of the Statistics Toolbox of Matlab. Specifically, it will be the libsvm library what will provide us the training and prediction algorithms that we need to develop our classification method, the support vector machines (SVM).

As we said before, there are many classifiers that also work properly with this kind of problems. However, we consider it would be more efficient and interesting to check the performance of the SVM running together the recursive feature elimination (RFE) algorithm. The object of using these two methods is because we are looking forward establishing a ranking of the genes in the sequence to test its relevance in certain diagnosis. As our sequences are composed of thousands elements we desire to reduce the dimensions of the array in order to extract those unrelated to the illness we are trying to classify.

Moreover we thought about implementing a new version of the RFE extraction method. With this algorithm, that we called robust RFE, we want to increment the accuracy in the sort and reduce the average error rate. It will be based on the repetition of the RFE about a thousand times. In each iteration we will modify the samples that are used in the classification so we will have a different ordination of the variables every time. Finally we will make an average of when are extracted the variables for the purpose of knowing its relevance.

Once we have concluded the classification process, we will establish a comparison between the current RFE algorithm and our robust RFE. The reason for this is to determine if the extra time we need to develop this new method helps us to obtain a higher success rate and a more conclusive ranking of the genes.

To sum up, as the number of patients we can work with is relatively small compared to the number of genes en each sequence, we need to select the most related to those pathologies we are studying. Our objective is to develop and test an algorithm that become able to extract accurately what are those conclusive genes in order to check later how the error rate changes if we use all the variables or only the most relevant ones.

CAPÍTULO 1: MÉTODOS DE CLASIFICACIÓN LINEAL

1.1 INTRODUCCIÓN

1.2 FISHER LINEAR DISCRIMINANT

1.3 PERCEPTRÓN

1.4 SUPPORT VECTOR MACHINES

1.1 INTRODUCCIÓN

En nuestro estudio disponemos de un conjunto de datos que representan un gran desafío ya que contienen un gran número de dimensiones (cerca de los diez mil elementos) de expresión genética por cada experimento y un número relativamente pequeño de experimentos (unos 100 o 200 aproximadamente). Esto significa que nuestro problema es separable, tenemos más grados de libertad de los que necesitamos, podríamos establecer infinitos umbrales de decisión y todos ellos serían correctos. En términos matemáticos podríamos decir que tenemos un problema con infinitas soluciones, ya que disponemos de más dimensiones que ecuaciones.

En estudios de este tipo es importante contemplar el problema de clasificación que tenemos que abordar. Tenemos a la entrada un vector NxD, donde a la componente N representará las muestras y la componente D, el espacio n-dimensional de muestras. En nuestro caso las muestras serían los fragmentos de secuencias de ADN de los pacientes y la segunda dimensión correspondería a los coeficientes de expresión genética.

La cuestión es que tenemos que encontrar maneras de reducir la dimensionalidad D del espacio de muestras con el objetivo de evitar el sobreajuste [1]. El sobreajuste de datos, que surge cuando el número de variables es muy grande (en nuestro caso miles de genes) y el número de patrones es relativamente pequeño en comparación (unos pocos pacientes), hace que el clasificador pierda capacidad de generalización. No obstante, en esta situación resulta sencillo encontrar una función de decisión que separe los datos de entrenamiento (es posible hacerse incluso con una función lineal).

Por este motivo, hemos decidido emplear un clasificador lineal capaz de separar correctamente las muestras en función de la clase a la que pertenecen. A pesar de que existen múltiples clasificadores lineales que resolverían correctamente nuestro problema, dado que nuestro objetivo no es clasificar sino realizar una selección de los genes que más información nos aportan a la hora de clasificar, queremos utilizar uno que asigne a cada variable de la muestra un peso en función de su relevancia. Para ello hemos seleccionado los tres que consideramos más útiles a la hora de establecer un ranking de las variables mediante la asignación de pesos a las mismas. Estos clasificadores son:

- Fisher linear discriminant (LDA).
- Algoritmo del perceptrón.
- Máquinas de vectores de soporte (Support Vector Machines, SVM).

1.2 DISCRIMINANTE LINEAL DE FISHER

El primer algoritmo de clasificación que vamos a estudiar es el basado en el análisis discriminante lineal, particularmente el discriminante lineal de Fisher [10], que llevará a cabo una clasificación lineal por medio de la reducción del espacio de dimensiones. Aunque el modelo de Fisher no puede ser considerado un discriminante en sentido estricto, resulta realmente útil cuando queremos construir uno.

Consideremos por ahora un modelo binario en el que sólo tenemos dos clases. Si tomásemos el vector de entrada (D-dimensional) y quisiéramos proyectarlo en una única dimensión obtendríamos un resultado de la forma:

$$\boldsymbol{y = w^T x};\quad (1.1)$$

Donde *w* representa el vector de pesos que tendremos que ajustar manualmente. Dado que en general la proyección de un vector de D-dimensiones sobre un espacio de una única dimensión conlleva una importante pérdida de información, y que las clases que en D-dimensiones estaban bien diferenciadas aquí pueden superponerse, resulta decisiva la elección de aquellos valores de *w* que den lugar a una menor pérdida de información. Esto se consigue ajustando las componentes del vector de pesos de tal forma que obtengamos la proyección que maximice la separación entre clases. La idea de Fisher es maximizar una función que nos proporcione una mayor separación entre ambas clases para evitar en la medida de lo posible la superposición de ambas.

Si en el modelo de dos clases mencionado anteriormente tuviéramos N_1 puntos de la clase C_1 y N_2 puntos de la clase C_2, de tal forma que las medias de ambas clases fueran:

$$\boldsymbol{m_1 = \frac{1}{N_1}\sum_{n\in C_1} x_n} \qquad \boldsymbol{m_2 = \frac{1}{N_2}\sum_{n\in C_2} x_n}\quad (1.2)$$

Podríamos realizar una simple separación de ambas clases mediante la separación de las medias de ambas clases proyectadas. Por ello, deberíamos escoger unos valores de *w* que maximizaran la función

$$\boldsymbol{m_2 - m_1 = w^T(m_2 - m_1)}\quad (1.3)$$

Donde

$$\boldsymbol{m_k = w^T m_k}\quad (1.4)$$

es la media de los datos proyectados de la clase C_k. Sin embargo, esta expresión podría crecer demasiado si escogiéramos valores de *w* excesivamente grandes, por ello limitaremos el valor de w de forma que $\sum_i w_i^2 = 1$. Utilizando un multiplicador de Lagrange para resolver esta maximización restringida encontramos que $w \propto (m_2 - m_1)$. Pero todavía tenemos problemas de muestras erróneamente clasificadas con esta aproximación. Lo que propone Fisher es encontrar esa

maximización entre clases pero con una mínima varianza en cada clase de forma que se redujera notablemente la superposición.

La fórmula $\boldsymbol{y = w^T x}$; (**1**.1) nos transformaba un conjunto de etiquetas de entrada del espacio x en un conjunto de etiquetas y de una única dimensión. La varianza de los datos transformados de una clase C_k viene dada por

$$\boldsymbol{s_k^2 = \sum_{n \in C_k} (y_n - m_k)^2} \quad (1.5)$$

Donde $y_n = w^T x_n$. Podemos definir la varianza del conjunto total de datos como $s_1^2 + s_2^2$. El criterio de Fisher se define como la relación que existe entre la varianza dentro de la propia clase y la varianza entre clases. Esto se puede expresar como

$$\boldsymbol{J(w) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}} \quad (1.6)$$

o lo que es los mismo

$$\boldsymbol{J(w) = \frac{w^T S_B w}{w^T S_W w}} \quad (1.7)$$

En esta fórmula, S_B representa la matriz de covarianza entre clases y S_W la matriz de covarianza dentro de la clase, que son obtenidas mediante las siguientes fórmulas:

$$\boldsymbol{S_B = (m_2 - m_1)(m_2 - m_1)^T} \quad (1.8)$$

$$\boldsymbol{S_W = \sum_{n \in C_1} (x_n - m_1)(x_n - m_1)^T + \sum_{n \in C_2} (x_n - m_2)(x_n - m_2)^T} \quad (1.9)$$

Si despejamos $\boldsymbol{J(w) = \frac{w^T S_B w}{w^T S_W w}}$ (1.**7**) con respecto a w encontramos que $J(w)$ se hace máxima cuando

$$\boldsymbol{(w^T S_B w) S_W w = (w^T S_W w) S_B w} \quad (1.10)$$

De la ecuación $\boldsymbol{S_B = (m_2 - m_1)(m_2 - m_1)^T}$ (1.**8**) podemos extraer que $S_B w$ tiene siempre la misma dirección que $(m_2 - m_1)$. Como lo que nos importa no es la magnitud de w sino su dirección, podemos eliminar los factores escalares $(w^T S_B w)$ y $(w^T S_W w)$. Multiplicando ambos lados de la ecuación de antes por S_W^{-1} obtenemos

$$\boldsymbol{w \propto S_W^{-1}(m_2 - m_1)} \quad (1.11)$$

Es importante remarcar que si la matriz de covarianza de clases es isotrópica, entonces S_W es proporcional a la matriz identidad y por tanto obtendremos unos valores que hagan que el vector w sea proporcional a la diferencia de medias de las clases.

Este resultado es lo que se conoce como discriminante lineal de Fisher. Aunque como dijimos anteriormente, no es estrictamente un discriminante, sino un algoritmo de elección de la dirección de la proyección de los datos a una sola dimensión. Sin embargo, los datos proyectados podrán ser posteriormente utilizados para construir un nuevo discriminante.

1.3 PERCEPTRÓN

Otro ejemplo de discriminante lineal es el algoritmo del perceptrón [10], que ocupa un papel fundamental entre los algoritmos de reconocimiento de patrones. El perceptrón simple se utiliza como un clasificador lineal en el que, dado un conjunto de muestras, se pretende encontrar el hiperplano capaz de separar esas muestras en dos clases.

El algoritmo del perceptrón es la forma más simple de red neuronal. Está constituido por una única neurona que será la que reciba las muestras de entrada. Cada conexión dentro de la neurona llevará asociados unos pesos, que junto al vector de entrada nos permitirán calcular la salida. Esta salida vendrá determinada, además de la información de las muestras de entrada y el vector de pesos, por un umbral y una función de activación, de forma que:

$$\boldsymbol{y = f(x_1 w_1 + x_2 w_2 + \cdots + x_n w_n + u)} \quad (1.12)$$

Donde y representa la salida, w los pesos, x las muestras de entrada, u el umbral y f la función de activación. A continuación presentamos una imagen que ilustra el funcionamiento del perceptrón simple. [4]

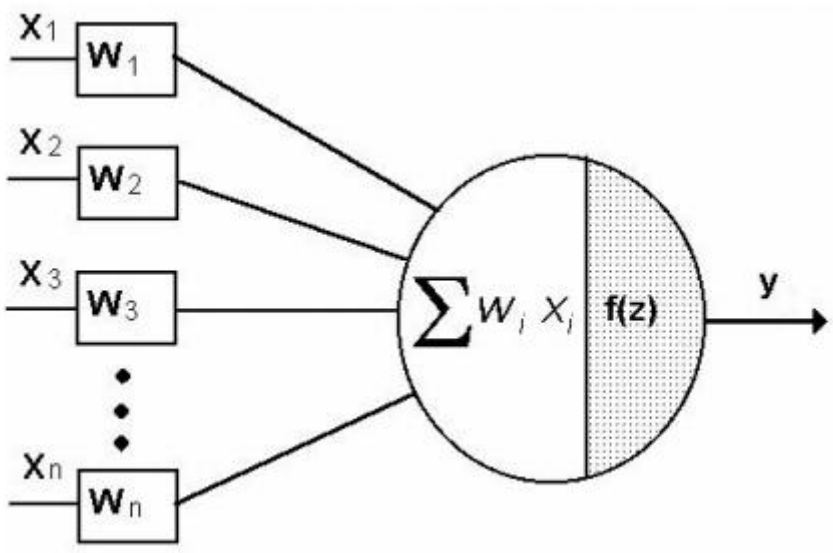


Ilustración 1.1. Arquitectura del perceptrón simple

Generalmente utilizaremos la función escalón como función de activación, cuya representación mostramos debajo.

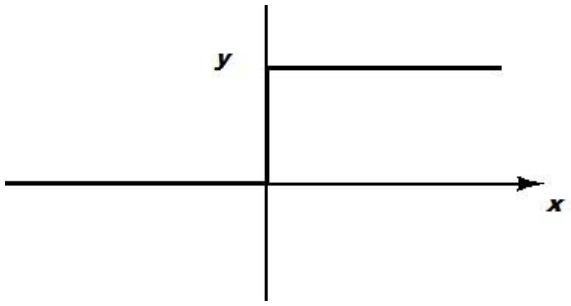


Ilustración 1.2. Función escalón como función de activación

El método de aprendizaje para el algoritmo del perceptrón es un proceso iterativo. Se basa fundamentalmente en la modificación de los parámetros del vector de pesos y en la realización de pequeñas variaciones del umbral con el propósito de encontrar el hiperplano discriminante, teniendo como condición de parada concluir un número determinado de iteraciones o la obtención de un resultado que se encuentre dentro de los parámetros de error definidos de antemano. Dado que no se trata de un algoritmo trivial, vamos a definir el proceso completo paso por paso.

En primer lugar inicializamos de forma aleatoria el vector de pesos y el valor del umbral, con lo que ya tendríamos $w_i(0)$ y $u(0)$. Además, debemos escoger un patrón de entrada y salida de la forma $[x = (x_1, x_2, \dots, x_n), d(x)]$, donde $d(x)$ representa la salida de deseada para ese patrón en concreto. A continuación, calcularíamos la salida de la red como definimos inicialmente:

$$\boldsymbol{y = f(x_1 w_1 + x_2 w_2 + \cdots + x_n w_n + u)} \quad (1.13)$$

Y es ahora cuando concluimos. Si $y = d(x)$, entonces la clasificación será correcta. Si por el contrario,$y \neq d(x)$ la clasificación será errónea y deberemos modificar los parámetros.

Este procedimiento lo repetiremos para todos los patrones entrada-salida que hayamos definido, y con cada patrón tantas veces como sean necesarias hasta que se cumpla la condición de parada.

Para definir la condición de parada podríamos utilizar el siguiente criterio. Tomamos el número de aciertos en una iteración del perceptrón, si este número es igual al número de aciertos de la iteración anterior, en ese caso aumentaremos las repeticiones, de manera que si se obtuvieran diez repeticiones se pararía la ejecución del perceptrón aunque no se hubiese alcanzado el número máximo de iteraciones. Si se diera el caso de que el número total de aciertos en una iteración fuera mayor al que tenemos almacenado, guardaríamos el valor de los pesos y del umbral para ese perceptrón y volveríamos a inicializar todos los parámetros.

Procediendo de esta manera, nos aseguramos que el perceptrón no vaya a ejecutar iteraciones innecesarias que nos ralentizarían la ejecución y que el resultado que

obtenemos será el mejor de todo el proceso de ejecución. Esto nos permitirá más adelante calcular los aciertos en test y validación con estos valores, obteniendo siempre el resultado óptimo dentro de todas las posibles iteraciones que se hayan ejecutado.

Por último es necesario definir la forma en que se van a modificar los parámetros, que se basa en las siguientes expresiones:

$$\boldsymbol{w_i(t+1) = w_i(t) + x_i d(x)} \quad \forall \boldsymbol{i} \quad (1.14) \qquad \boldsymbol{u(t+1) = u(t) + d(x)} \quad (1.15)$$

Esto se conoce como Ley de aprendizaje, y aunque nos proporcionaría un resultado óptimo desde el punto de vista del tiempo de ejecución, podrían producirse pequeños errores a la hora de adaptar los pesos. Por este motivo, es importante mencionar también la regla de aprendizaje de Widrow-Hoff, que tiene un comportamiento similar a la ley anterior, pero que se basa en la idea de utilizar el error cometido por la red neuronal para adaptar mejor los valores de los pesos:

$$\boldsymbol{w_i(t+1) = w_i(t) + [x_i d(x) - y(x)]} \quad \forall \boldsymbol{i} \quad (1.16)$$

$$\boldsymbol{u(t+1) = u(t) + [d(x) - y(x)]} \quad (1.17)$$

Otro procedimiento también válido sería utilizar la razón de aprendizaje. Esto evitaría que los nuevos parámetros clasificaran de forma incorrecta patrones que anteriormente habían sido correctamente clasificados. Si tomamos como razón de aprendizaje α, donde $0 < \alpha < 1$:

$$\boldsymbol{w_i(t+1) = w_i(t) + \alpha d(x) x_i} \quad \forall \boldsymbol{i} \quad (1.18) \qquad \boldsymbol{u(t+1) = u(t) + \alpha d(x)} \quad (1.19)$$

El motivo de incluir este nuevo parámetro es controlar la brusquedad con la que se podrían modificar el resto de parámetros. Una última opción sería utilizar la regla de aprendizaje de Widrow-Hoff incorporando la razón de aprendizaje.

1.4 SUPPORT VECTOR MACHINES (SVM)

Cuando tenemos un problema linealmente separable en el que existen varias soluciones para separar los datos en clases, podemos utilizar diversos algoritmos. Con el algoritmo del perceptrón, por ejemplo, garantizábamos encontrar una solución en un número finito de iteraciones. Sin embargo, la solución encontrada dependía de los valores escogidos inicialmente para w y u, así como el orden en el que presentáramos los datos.

Si tenemos múltiples soluciones que nos clasifican los datos correctamente, lo que podemos hacer es buscar la solución que nos proporcione el error mínimo. Las SVM [1] hacen una aproximación de este problema a través del concepto de margen, que se define como la menor distancia entre el límite de decisión y cualquiera de las muestras. Además nos proporcionan una gran versatilidad y unas prestaciones enormemente favorables. Este hecho hace que actualmente tengan un elevado número de aplicaciones en campos como el reconocimiento de textos y escrituras, recuperación de información, clasificación de imágenes…

Ahora vamos a proceder a describir el funcionamiento del algoritmo de las máquinas de vectores de soporte.

Antes de comenzar a clasificar tenemos que llevar a cabo un procedimiento previo de aprendizaje. En esta etapa lo que haremos será buscar el hiperplano $h(x) = 0$ que nos separe de forma óptima el conjunto de datos de entrada $X \in \mathbb{R}^d$ en función de la clase $Y \in \{-1,1\}$ a la que pertenecen. Este hiperplano será el que maximice la distancia al punto más próximo de cada clase, por ello, estará a la misma distancia de los datos más cercanos de cada categoría.

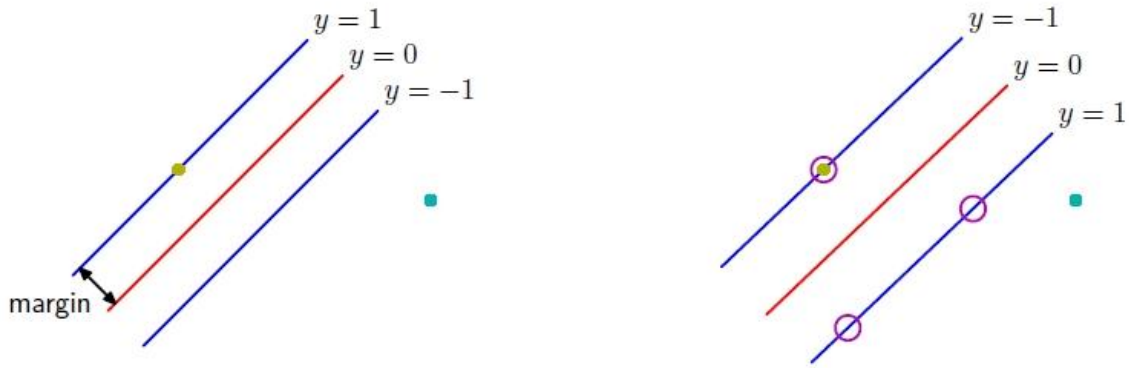


Ilustración 1.3. Margen en clasificador SVM. El margen se define como la distancia entre el límite de decisión y la muestra más cercana, tal y como se muestra en la figura de la izquierda. Maximizar el margen exige escoger un determinado límite de decisión. En la figura de la derecha se muestra como la elección de este límite viene determinado por un subconjunto de muestras conocido como vectores de soporte, en la figura son las que aparecen rodeadas.

Según la formulación original del algoritmo, el separador que maximizará el margen será aquel que nos dé una capacidad mayor para localizar características comunes de las muestras de una misma clase con el objetivo posterior de ser capaces de clasificar

correctamente muestras que no pertenezcan al conjunto de entrenamiento. Para obtener ese separador deberemos resolver primero un problema de optimización en el que utilizaremos técnicas de programación cuadrática.

Los datos empleados para encontrar el hiperplano, que constituirá nuestra frontera de decisión, serán los vectores de aprendizaje o vectores de entrenamiento. Estos vectores nos permitirán crear las estructuras o modelos de clasificación con los que trabaja la SVM para clasificar los datos de test (muestras nuevas que no hemos utilizado para entrenar la máquina).

A partir de unos datos de entrada x_i, las SVM nos devolverán su clase según la regla de clasificación $f(x_i) = signo\ (h(x_i))$. En la figura de abajo aparecen datos de dos clases separados por el umbral de decisión que maximiza la distancia entre ambos. Esta distancia máxima es el margen que hemos obtenido para el hiperplano escogido. Si utilizáramos otro hiperplano distinto al que nos proporciona la SVM el margen, y por tanto la separación entre clases, sería menor.

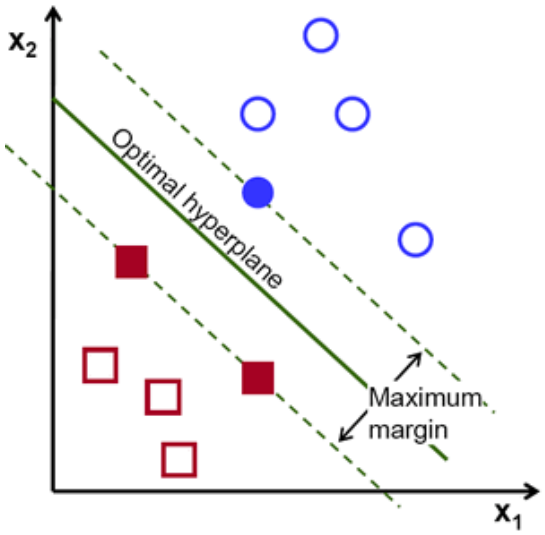


Ilustración 1.4. Hiperplano de separación óptimo normalizado para conjuntos linealmente separables.

Una vez finalizado el período de aprendizaje se comprueba la tasa de aciertos (o de error) tomando una nueva muestra en la que compararemos la salida que nos proporcione el clasificador con la clase a la que verdaderamente pertenece. Del conjunto total de datos lo que se suele hacer es tomar aproximadamente un 75% de las muestras como datos de entrenamiento y dejar el 25% restante como datos de test para contrastar la fiabilidad de la frontera de decisión escogida.

Este último paso resulta fundamental para que los modelos obtenidos puedan ser utilizados de manera fiable y aplicados a cualquier conjunto de datos posterior.

CAPÍTULO 2: MÁQUINAS DE VECTORES DE SOPORTE (SVM)

2.1 INTRODUCCIÓN

2.2 DESCRIPCIÓN DE LAS SVM

2.3 SVM LINEAL BINARIO

2.4 SVM LINEAL MULTICLASE

2.1 INTRODUCCIÓN

Una vez definidos los modelos anteriores de clasificación, procederemos a describir en profundidad en el SVM, que por sus altas prestaciones y tasas de acierto, además de por el tipo de datos que vamos a utilizar (y que más adelante expondremos), será el que emplearemos a la hora de efectuar la clasificación en los arrays de genes.

Aunque estas máquinas de vectores de soporte son capaces de trabajar con límites de decisión no lineales (y de complejidad arbitraria), en este trabajo nos limitaremos a una SVM lineal debido a la naturaleza de los datos que se investigan y al propósito de nuestro estudio, que es determinar la relevancia de cada una de las variables de nuestras muestras.

Actualmente disponemos de una serie de secuencias genéticas que constan, cada una, de cadenas de miles de genes. Cada uno de estos genes nos aporta individualmente cierta información acerca de un determinado sujeto. Sin embargo, nuestro objetivo es establecer un orden o determinar cuáles de estos nos van a resultar relevantes a la hora de concluir un diagnóstico, es decir, en qué posiciones dentro de la secuencia genética nos tenemos que fijar a la hora de decidir si en un paciente se manifestará una determinada patología o no. Dado que existen múltiples formas de llevar a cabo la clasificación (y todas ellas serían correctas), se busca encontrar los umbrales óptimos de decisión sin caer en el sobreajuste.

Puesto que este tipo de conjuntos de datos es linealmente separable, resulta lógico utilizar una SVM lineal que nos supondrá un importante ahorro desde el punto de vista del coste computacional y del desarrollo del algoritmo. Además nos proporcionará una clasificación óptima con el máximo margen entre clases. El límite de decisión estará posicionado para dejar la mayor distancia posible entre muestras de diferentes clases.

Por otra parte, resulta necesario mencionar que no sólo utilizaremos las máquinas de vectores de soporte para llevar a cabo la clasificación de las secuencias genéticas. También utilizaremos como punto de partida el algoritmo de eliminación recursiva de muestras (RFE) con el objetivo de ir reduciendo el número de dimensiones de nuestro problema y de esta manera poder quedarnos con las posiciones dentro del array que nos proporcionen mayor cantidad de información a la hora de clasificar.

2.2 DESCRIPCIÓN DE LAS SVM

En el primer capítulo de nuestro estudio introdujimos brevemente el funcionamiento general de las máquinas de vectores de soporte (SVM). En este apartado, sin embargo, vamos a analizar detalladamente la manera de proceder de este algoritmo.

Dados los vectores de entrenamiento $x_i \in R^n, i = 1, \dots, l,$ de dos clases, y un vector de etiquetas $y \in R^l$, de manera que $y_i \in \{-1,1\}$, suponemos que tenemos un hiperplano que nos separa las muestras positivas y negativas. Los puntos de x que yacen en el plano serán los que satisfagan la ecuación $w \cdot x + b = 0$, donde w es ortonormal al hiperplano, $|b|/\|w\|$ es la distancia perpendicular desde el hiperplano al origen, y $\|w\|$ es la norma de w. Si consideramos d_+ (o d_-) como la distancia más corta desde el hiperplano a la muestra positiva (o negativa) más cercana y definimos el margen como $d_+ + d_-$, para el caso lineal separable, el algoritmo SVM buscará el hiperplano que nos ofrezca un margen mayor. Esto puede ser formulado de la siguiente forma.

Suponiendo que todas las muestras de entrenamiento satisfacen las siguientes restricciones,

$$\boldsymbol{x_i \cdot w + b \geq +1 \quad para \ y_i = +1} \text{ (20.1)}$$

$$\boldsymbol{x_i \cdot w + b \leq -1 \quad para \ y_i = -1} \text{ (21.2)}$$

que combinadas dan lugar a la siguiente inecuación

$$\boldsymbol{y_i(x_i \cdot w + b) - 1 \geq 0 \quad \forall i} \text{ (223)}$$

Los puntos en los que se cumpla que $x_i \cdot w + b = 1$ formarán el hiperplano H_1, con norma w y distancia perpendicular al origen $|1 - b|/\|w\|$. De manera similar, los puntos que cumplan $x_i \cdot w + b = -1$ constituirán el hiperplano H_2, con la misma norma que H_1 y distancia perpendicular al origen $|-1 - b|/\|w\|$. Por tanto $d_+ = d_- = 1$ y el margen es simplemente $2/\|w\|$. Nótese que H_1 y H_2 van a ser paralelos y que no existirá ninguna muestra entre ellos. La tarea es encontrar entonces aquel par de hiperplanos que dé el máximo margen minimizando $\|w\|^2$.

En la figura de abajo se ilustra cómo sería la solución para este caso. Aquellas muestras de entrenamiento que satisficieran la ecuación anterior y cuya eliminación provocaría un cambio en la solución serán los llamados vectores de soporte (en la imagen aparecen rodeados).

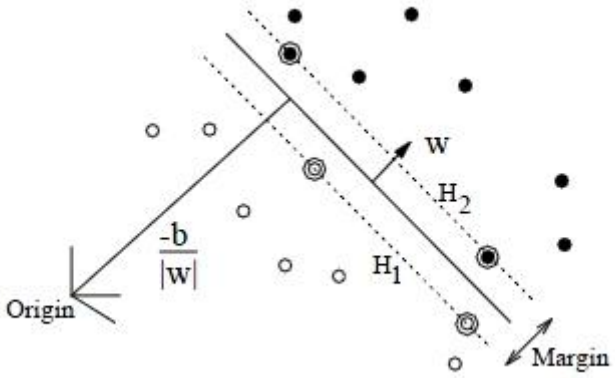


Ilustración 2. 1 Hiperplanos lineales para el caso separable. Los vectores de soporte aparecen rodeados.

No obstante, en la implementación de este algoritmo se realiza una formulación de Lagrange, puesto que lo que necesitamos es resolver una optimización con restricciones. Al introducir los multiplicadores de Lagrange $\alpha_i, i = 1, \dots, l$, obtendremos una expresión de la forma

$$\boldsymbol{L_P \equiv \frac{1}{2}\,\|w\|^2 - \sum_{i=1}^{l} \alpha_i y_i (x_i \cdot w + b) + b + \sum_{i=1}^{l} \alpha_i} \quad (23)$$

Ahora se trata de minimizar L_P con respecto a $\boldsymbol{w}$, b y además exigir que las derivadas de L_P respecto a todos los coeficientes α_i estén sujetas al requisito $\alpha_i \geq 0$. Nos enfrentamos por tanto a un problema cuadrático convexo, ya que estamos utilizando una función convexa y el conjunto de puntos que satisfagan los requisitos será también convexo. Esto significa que tenemos que resolver el siguiente problema dual: hay que maximizar L_P teniendo en cuenta el gradiente de L_P respecto a $\boldsymbol{w}$ y b pero también que los coeficientes α_i tienen que tomar valores positivos. De esta forma tenemos las siguientes condiciones:

$$\boldsymbol{w = \sum_i \alpha_i y_i x_i} \quad (24)$$

$$\boldsymbol{\sum_i \alpha_i\, y_i = 0} \quad (25)$$

que si las sustituimos en $\boldsymbol{L_P \equiv \frac{1}{2}\,\|w\|^2 - \sum_{i=1}^{l} \alpha_i y_i (x_i \cdot w + b) + b + \sum_{i=1}^{l} \alpha_i}$ **(23)** obtenemos

$$\boldsymbol{L_D \equiv \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)} \quad (26)$$

Nótese que hemos utilizado etiquetas distintas para las ecuaciones de Lagrange (P por principal, D por dual) con el objetivo de enfatizar que las dos formulaciones son diferentes: L_D y L_P surgen a raíz del mismo problema pero están sujetas a requisitos diferentes, y la solución es encontrada minimizando L_P o maximizando L_D.

Las máquinas de vectores de soporte creadas por el algoritmo maximizarán L_D siempre bajo los requisitos definidos anteriormente. Tendremos un multiplicador de Lagrange α_i por cada muestra de entrenamiento y en la solución sólo aquellos elementos que

cumplan $\alpha_i > 0$ serán vectores de soporte. Todas las demás observaciones tendrán un $\alpha_i = 0$ y deberán ubicarse en su espacio correspondiente.

2.3 SVM LINEAL BINARIO

Aunque para llevar a cabo la resolución de nuestro problema necesitaremos una SVM lineal para varias clases, se ha considerado de gran utilidad la introducción de una SVM con únicamente dos clases con el objetivo de demostrar de manera empírica su funcionamiento de un modo óptimo. Asimismo es importante mencionar en este punto que para la implementación de estos algoritmos nos basaremos en la librería *libsvm* (APÉNDICE C: LIBSVM) de Matlab.

Para este ejemplo se han generado conjuntos de datos de forma aleatoria. La finalidad de este *script*, llamado *GeneraDatos*, es ilustrar el comportamiento de las máquinas de vectores de soporte cuando se tienen múltiples muestras pero únicamente dos dimensiones. Esto es debido a que el hecho de que sea un problema bidimensional nos va a ayudar a comprender mejor el proceso de clasificación de las SVM.

Al final de la ejecución de este código tendremos las variables:

- *X_entrenamiento*: se trata de una matriz con tantas filas como muestras disponemos (200), y tantas columnas como dimensiones o clases presenta el problema (para este ejemplo dos). Será utilizada para entrenar la SVM.
- *Y_entrenamiento*: es una matriz con el mismo número de filas que la matriz anterior, pero con una única columna. En esta matriz se encuentran almacenadas las clases a las que debe pertenecer cada elemento de la *X_entrenamiento*. Se utilizará para comprobar durante el período de entrenamiento si la clasificación se está realizando de manera correcta.
- *X_test*: una vez que haya finalizado la etapa de entrenamiento, emplearemos los datos aquí almacenados para comprobar si todo el proceso realizado anteriormente funciona correctamente con datos nuevos.
- *Y_test*: al igual que sucedía con *Y_entrenamiento*, esta matriz será empleada para comprobar, una vez concluido el entrenamiento con las muestras, si los clasificadores que se han construido son capaz de decidir a qué clase pertenecen muestras que nunca antes han visto.

Una vez definidos los datos, comenzaremos con el algoritmo de las SVM. Como ahora no disponemos de datos de entrenamiento y de test sino que tenemos una matriz con observaciones y otra con etiquetas, hemos decidido probar a utilizar la técnica *leave-one-out* para llevar a cabo el entrenamiento de las SVM. Esta técnica se basa en la existencia de un único conjunto de datos a partir del cual, siguiendo un criterio establecido por nosotros mismos, utilizaremos una parte de esos datos como datos de test, y el resto como conjunto de entrenamiento. Esta operación de extracción de los

datos la realizaremos tantas veces como muestras distintas tengamos, es decir, tantas veces como filas tenga la matriz con los datos de entrada (las columnas de esta matriz representarán las dimensiones de los datos a analizar). En cada iteración seleccionaremos un valor distinto a utilizar como dato de test y anotaremos para cada caso si con ese elemento hemos obtenido un acierto o un fallo. Una vez hayamos concluido podremos promediar los resultados para determinar el porcentaje de error que obtuvimos con esa selección.

En este primer SVM binario además hemos incluido un vector en el que introducimos tres valores distintos para el parámetro de coste: 10, 100 y 1000. La finalidad de la incorporación de este vector es la realización de un bucle que constará únicamente de veinte iteraciones y que nos permitirá determinar cuál de los valores de coste escogidos anteriormente nos proporciona un resultado con una tasa de aciertos óptima en un tiempo de ejecución aceptable.

La elección de un parámetro de coste adecuado es importante en este tipo de problemas en los que tenemos más dimensiones que muestras. Si realizamos una analogía con los sistemas de ecuaciones, podríamos decir que nuestro problema es un sistema compatible indeterminado donde cada muestra representa a una ecuación y los pesos que nos proporcione el clasificador lineal serán las incógnitas. Dado que este problema es separable (tenemos más grados de libertad de los que necesitamos), existirán infinitos planos que separen los datos sin errores y escoger un valor óptimo para este parámetro nos proporcionará cierta mejora en los resultados finales.

Seleccionar un valor adecuado del coste, C, viene estrechamente relacionado con los coeficientes alfa que nos proporcionaba la *libsvm (*APÉNDICE C: LIBSVM*)*. Estos coeficientes surgen a raíz del problema cuadrático de optimización de las SVM que comentamos al inicio del capítulo.

Los valores que tomen estos coeficientes α_i, que representan la contribución de cada muestra de entrenamiento al cálculo del valor final de w, pueden venir agrupados en tres tipos:

- $\alpha = 0$: todas las muestras quedaron bien clasificadas dentro del margen.
- $0 < \alpha < C$: las muestras se hallan alrededor del margen.
- $\alpha = C$: las muestras están mal clasificadas. Como el problema no es separable, algunas tienen que estar en el lado incorrecto del plano.

Si el problema es separable, no existirán muestras en las que $\alpha = C$. Si escogiéramos un valor de C en el que esto se cumpliera, deberíamos cambiarlo y volver a entrenar la SVM. La idea es utilizar el mínimo valor de C con el que todos los valores de α sean menores a él y a partir del cual, un incremento de C no supondría una mejora de los resultados.

A continuación vamos a realizar una explicación paso a paso del código implementado para el algoritmo de clasificación de las SVM para un caso binario, con la finalidad de analizar de manera más detallada su desarrollo.

Antes de nada, invocamos al script *GeneraDatos* para que nos cargue las matrices de entrenamiento y test mencionadas anteriormente. Además, incorporamos un vector que contenga los parámetros que hemos decidido asignar inicialmente (10, 100 y 1000) para determinar el valor óptimo del coste.

A continuación, creamos un bucle que nos va a recorrer todas las filas del conjunto de muestras. Dentro de ese bucle lo primero que hacemos es seleccionar cuál va a ser la muestra escogida como test, y dejamos el resto como muestras de entrenamiento. Seguidamente introducimos una matriz llamada *aciertos*, de tantas filas como pruebas vamos a hacer para determinar qué valor del coste resulta más adecuado para nuestro problema, y tantas columnas como valores habíamos escogido inicialmente.

Dentro del bucle del *leave-one-out* nos encontramos, en primer lugar, con el bucle del valor del coste, que se ejecutará tantas veces como opciones escojamos a la hora de declararlo. La finalidad de este bucle es recoger, para cada columna de la matriz *aciertos*, qué valor inicial de coste nos proporcionará una mayor tasa de éxito. Para realizar esta comprobación de manera exacta habría que volver a repetir aquí otro bucle con tantas iteraciones como número de muestras, sin embargo, dado que es un ejemplo con finalidad didáctica, lo repetiremos únicamente veinte veces.

Para este nuevo *for* hemos incorporado un nuevo vector de longitud veinte, que tomará valores aleatorios dentro de las posibles posiciones de la matriz de muestras. Una vez más volvemos a seleccionar qué índice utilizaremos para test y cuáles dejaremos para entrenamiento. Llegados a este punto nos servimos de las funciones *svmtrain* y *svmpredict* para realizar la clasificación, en la que obtendremos una etiqueta que nos dirá la clase a la que el clasificador estima que pertenece esa muestra, y que utilizaremos para contrastar si su valor coincide con el valor de *Y_entrenamiento* en la posición del dato que estamos utilizando para test. En caso afirmativo, lo anotaremos como acierto (1) y en caso negativo como fallo (-1 o 0).

Cuando hayamos acabado con todos los valores que podíamos extraer del conjunto restante de entrenamiento ya seremos capaces de determinar cuál es el valor óptimo del coste a partir de la matriz *aciertos* que hemos ido rellenando en las sucesivas iteraciones. Para ello emplearemos la función mean que nos determine la media de cada una de las columnas de la matriz, y almacenaremos cuál es la que nos proporciona una tasa mayor. Entonces ya podemos escoger un único valor para el coste. Con este valor ya podremos volver a utilizar *svmtrain* y *svmpredict*, pero esta vez con todos los valores de entrenamiento y el valor de test que habíamos seleccionado en un principio. Y será a partir de este resultado cuándo podremos determinar la tasa final de acierto.

2.4 SVM LINEAL MULTICLASE

Todo lo que habíamos expuesto hasta ahora quedaba referido a un problema en el que los datos podían pertenecer únicamente a dos clases distintas. Sin embargo, el caso que se plantea en este trabajo parte de una serie de secuencias genéticas en las que los elementos a clasificar pueden pertenecer a once clases diferentes.

Aunque seguiremos utilizando el método *leave-one-out* para llevar a cabo el entrenamiento de las máquinas SVM, este va a ser formulado de una manera ligeramente diferente para el caso multiclase. Asimismo, cabe mencionar que actualmente existen dos maneras distintas de implementar este algoritmo *leave-one-out* cuando tratamos con datos de entrada en los que el número de dimensiones es mucho mayor al número de muestras y, que además pertenecen a múltiples clases. Estos dos métodos son:

- *One-vs-all*: con este algoritmo partimos de la existencia de un conjunto de datos pertenecientes a m clases diferentes. En este caso, deberíamos entrenar m máquinas SVM diferentes. La SVM m marcaría como positivas las muestras pertenecientes a esa clase m, y como muestras negativas las muestras correspondientes a las $m - 1$ clases restantes. Una vez finalizado el período de entrenamiento lo que haríamos es tomar una de las muestras escogidas como test y evaluarla para todas las m máquinas SVM. Aquella que nos proporcionara una salida blanda más grande, es decir, aquella que nos devolviera una distancia mayor respecto al margen definido, sería la que se asignaría como clase correspondiente para la muestra de test tomada, ya que cuanto más lejos está la muestra de la frontera, más seguros podemos estar de que la clasificación se ha realizado correctamente. Esto es porque cada máquina se entrena para responder a la pregunta: *¿esta muestra pertenece a mi clase o no?* Si la salida es positiva quiere decir que la máquina en cuestión reclama la muestra para su clase. Si se diera una situación en la que tenemos un empate y varias máquinas reclamasen la muestra para su clase, esta muestra se asignaría a aquella clase en la que la salida blanda fuese mayor. Por el contrario, si una máquina nos proporciona una salida negativa podemos concluir que esa muestra no pertenece a su clase. Si ocurriera que todas las máquinas nos devolvieran una salida negativa, este empate se resolvería asignando esa muestra a la clase con una salida menos negativa, esto es, a aquella con una salida blanda menor en la que la muestra estuviera más cerca a su margen y, por tanto, estuviese menos segura de que la muestra no es de su clase.

- *One-vs-one*: el procedimiento que vamos a explicar a continuación realiza el proceso de entrenamiento a partir de parejas de clases. Por ejemplo, si tuviéramos un problema con cinco clases distintas (C_1, C_2, C_3, C_4, C_5) tendríamos que entrenar diez máquinas:

$$C_1 \; vs \; C_3$$

$$C_1 \; vs \; C_4$$

$$C_1 \; vs \; C_5$$

$$C_2 \; vs \; C_3$$

$$C_2 \; vs \; C_4$$

$$C_2 \; vs \; C_5$$

$$C_3 \; vs \; C_4$$

$$C_3 \; vs \; C_5$$

$$C_4 \; vs \; C_5$$

En cada máquina que entrenamos una clase será la positiva y otra la negativa. Será imprescindible saber cuál es cada una cuando estemos nos encontremos en el proceso de entrenamiento para luego realizar el test correctamente. Para poder determinar a qué clase pertenece cada muestra de test una vez concluido el entrenamiento deberemos evaluar todas las máquinas de las que disponemos. Por cada muestra tendremos entonces diez "votos", uno por cada máquina, que se utilizarán para determinar la clase a la que pertenece esa muestra. Dada la forma en que se ha establecido el reparto de las clases en las diferentes máquinas, cada clase podrá recibir un máximo de cuatro votos (cada clase interviene en $m - 1$ máquinas). A la muestra en cuestión se le asignará la clase que haya recibido más votos. En caso de empate, una posible forma de proceder sería asignarle a la muestra la clase con la salida blanda más elevada. Otra forma un poco más eficiente sería ordenar las muestras en función de las clases y así volver a entrenar únicamente las máquinas de las clases en las que no interviene la muestra de test.

Puesto que para la resolución de nuestro problema nos servimos de la *Statistics Toolbox* de *Matlab*, y más concretamente de la librería *libsvm* (APÉNDICE C: LIBSVM), utilizaremos el método *One-vs-one* que es el que por defecto incorporan las funciones

svmtrain y *svmpredict* que necesitamos para llevar a cabo el entrenamiento y la clasificación de nuestras muestras.

A pesar de que este procedimiento de clasificación para las SVM multiclase es realizado internamente por el programa, ha sido necesaria la reescritura del código para poder adaptar los elementos de los que disponemos a la salida de las funciones, que en este caso será diferente a cuando teníamos únicamente dos clases. Por ello, vamos a explicar a continuación la implementación del algoritmo que hemos desarrollado para la clasificación de datos perteneciente a varias a clases.

En primer lugar es necesario hacer referencia al tipo de datos que se van a utilizar. En este caso no serán elementos generados por nosotros mismos sino que se trata de conjuntos reales almacenados en bases datos, extrapolables al problema que estudiaremos más adelante, en los que existen más dimensiones que número de observaciones. Puesto que de momento sólo queremos verificar el correcto funcionamiento de las máquinas de vectores de soporte, emplearemos únicamente uno de los datasets de los que disponemos, el conjunto *Carcinom*. Estos datos que utilizaremos vienen agrupados en dos matrices distintas, la matriz L (*de dimensiones 174x1*), para las etiquetas, y la matriz M (*con dimensiones 174x9182*) que contiene las muestras que se utilizarán para llevar a cabo el entrenamiento.

En la primera parte del código el procedimiento es bastante similar al que empleábamos cuando pretendíamos implementar una SVM binaria. Cabe destacar la incorporación de ciertos matices con respecto a aquel código. Aunque vamos a seguir utilizando los datos de la matriz M como muestras de entrenamiento, ahora calculamos una matriz de *kernels*, a partir del producto de M por su traspuesta. Asimismo, una vez declaradas las variables necesarias comenzamos el bucle *leave-one-out* donde definimos qué muestras van a ser utilizadas para entrenamiento y cuál será la empleada para realizar el test. Por último declaramos, igual que hacíamos antes, la matriz *aciertos*, donde iremos almacenando la tasa de acierto para un número aleatorio de muestras con cada valor del coste.

Llegados a este punto, lo que hacemos es iniciar el bucle para obtener el valor óptimo del coste. Aquí observamos otra diferencia respecto a la SVM binaria. Cuando declaramos las *options* que emplearemos en la clasificación, a la variable *t* no le asignamos el valor 0 como hacíamos anteriormente. Como vamos a clasificar con la matriz de kernels en lugar de con M (donde estaban almacenadas todas las muestras desde el principio) es necesario que este parámetro tenga el valor 4.

A continuación procedemos de nuevo a introducir un vector de índices al azar con la finalidad de que el bucle que utilizábamos para extraer el valor óptimo del coste tome un carácter aleatorio. Lo que se ejecuta en las primeras líneas de este bucle *for* vuelve a ser de nuevo el algoritmo *leave-one-out*, ahora para seleccionar, dentro del conjunto que ya teníamos de entrenamiento, un nuevo índice para realizar el test. Sin embargo,

inmediatamente después incorporamos lo novedoso de la clasificación multiclase. La salida de *svmpredict* ya no va a ser únicamente la etiqueta que se le asigna a esa muestra, ahora nos devolverá tres datos relativos a cada muestra: la etiqueta asignada, la precisión o accuracy de la clasificación y la distancia al margen establecido o salida blanda. De momento sólo utilizaremos la salida *etiqueta*, con el objetivo de comprobar si hemos cometido un fallo o hemos acertado, y de esta forma ir rellenando la matriz de aciertos para obtener el valor de coste que mejor se adapta a nuestros datos.

Para terminar lo que hacemos es extraer el parámetro del coste óptimo como valor de C definitivo y realizamos el entrenamiento de la máquina SVM y la comprobación con la matriz de etiquetas (L). De nuevo aquí la función *svmpredict* nos devuelve tres salidas, que aunque de momento no han sido utilizadas prácticamente, resultarán de gran utilidad en un futuro.

Una vez que conocemos el valor de coste óptimo que necesitamos para resolver nuestro problema, volvemos a reescribir el código de las SVM multiclase, pero esta vez eliminando el bucle que utilizábamos para calcularlo e incluyendo este valor como una variable más. Será a partir de esta implementación como desarrollaremos el algoritmo que nos permita seleccionar los genes más relevantes de las muestras de secuencias genéticas, que es el objetivo primordial en nuestro estudio.

CAPÍTULO 3: RECURSIVE FEATURE ELIMINATION (RFE)

3.1 INTRODUCCIÓN

3.2 RFE ESTÁNDAR

3.3 RFE ROBUSTO

3.1 INTRODUCCIÓN

Una vez que hemos comprobado el adecuado funcionamiento de las SVM para este tipo de problemas, es conveniente volver a realizar una importante anotación. El objetivo primordial de este proyecto no es solamente clasificar, sino realizar una selección de qué posiciones dentro de la secuencia genética son las más relevantes.

Para ello, vamos a utilizar como punto de partida el RFE (*recursive feature elimination*), un método propuesto por Isabelle Guyón en 2002 [1], para posteriormente desarrollar la implementación de una variante más robusta de este algoritmo que, ejecutado durante un elevado número de iteraciones cada una de ellas con unas muestras de entrenamiento distintas, nos permita obtener resultados más concluyentes acerca de la relevancia real de los genes disponibles en cada secuencia y su influencia para las distintas patologías.

En este desarrollo utilizaremos todos los conjuntos de datos de los que disponemos. De esta forma podremos comparar las soluciones que nos ofrecen ambos algoritmos con el fin de determinar que el algoritmo robusto, a pesar de tener mayor coste computacional nos ofrece un ordenamiento de las variables más fiable. Puesto que lo que más nos importa es ser capaces de seleccionar cuáles son los genes que nos permiten detectar de manera más eficiente una enfermedad en concreto, el tiempo de ejecución no nos va a suponer un grave problema si finalmente el *RFE* robusto nos ayuda a extraer esos genes.

3.2 RFE

El origen de este método es la búsqueda de un criterio que permita extraer variables relevantes dentro de un reducido conjunto de datos clínicos con un número muy elevado de variables. Se trata de eliminar aquellos elementos que resultan irrelevantes y así quedarnos con aquellos que afectan de manera directa a una patología, estableciendo ciertos patrones a la hora de realizar su diagnóstico.

RFE establece un criterio de clasificación de subconjuntos de muestras a partir de las SVM [1]. Se basa en tres pasos principales:

1. Entrenar el clasificador. Asignar los pesos w_i a todas las dimensiones de cada uno de los vectores de soporte.

2. Establecer un promedio del peso de cada variable para todos los vectores de soporte.

3. Eliminar las variables con menor peso en la clasificación.

SVM RFE es la aplicación de RFE que utiliza el vector de pesos w_i como criterio para establecer el orden de eliminación de las muestras. A continuación presentamos cómo se desarrollaría este algoritmo para el caso lineal.

Algoritmo SVM-RFE

Variables de entrada

Muestras de entrenamiento $\boldsymbol{X_0 = [x_1, x_2, \dots x_k, \dots x_l]^T}$ *, cada* x_i *será un vector de tantas columnas como genes (o variables) tenga esa secuencia (muestra).*

Etiquetas $\boldsymbol{y = [y_1, y_2, \dots y_k, \dots y_l]^T}$

Inicializamos

Conjunto de variables restantes
$\boldsymbol{s} = [\mathbf{1}, \mathbf{2}, \ldots \boldsymbol{n}]$
Lista de variables ordenadas por relevancia
$\boldsymbol{r} = []$

Repetir hasta que nos quedemos con menos variables de las que deseamos extraer
$\boldsymbol{s} = []$

Entrenamos el clasificador

$$\boldsymbol{\alpha} = \boldsymbol{entrenamiento}\ (\boldsymbol{X}, \boldsymbol{y}) \quad \textbf{(27)}$$

Restringimos las variables de entrenamiento a los índices seleccionados y nos quedamos sólo con las muestras que sean vectores de soporte.

$$\boldsymbol{X} = \boldsymbol{X_0}(:, \boldsymbol{s}) \quad \textbf{28}$$

Calculamos los valores del vector de pesos

$$\boldsymbol{w} = \sum_k \boldsymbol{\alpha_k y_k x_k} \quad \textbf{29}$$

Calculamos el criterio para la ordenación

$$\boldsymbol{c_i} = \boldsymbol{w_i} \quad , \quad \boldsymbol{para\ todo\ i} \quad \textbf{(30}$$

Buscamos la variable menos relevante

$$\boldsymbol{f} = \boldsymbol{argmin}(\boldsymbol{c}) \quad \textbf{(31}$$

Actualizamos la lista de ordenación

$$\boldsymbol{r} = [\boldsymbol{s}(\boldsymbol{f}), \boldsymbol{r}] \quad \textbf{(32}$$

Eliminamos las variables menos relevantes

$$\boldsymbol{s} = \boldsymbol{s}(\mathbf{1}: \boldsymbol{f} - \mathbf{1}, \boldsymbol{f} + \mathbf{1}: \boldsymbol{length}(\boldsymbol{s})) \quad \textbf{(33}$$

Salida

Lista ordenada de muestras $\boldsymbol{r}$

Este procedimiento iterativo es un ejemplo de una función de eliminación hacia atrás. Por razones de coste computacional, resulta mucho más eficiente eliminar varias dimensiones a la vez, en lugar de una a una como describíamos anteriormente (cuando utilizábamos $\boldsymbol{f} = \boldsymbol{argmin}(\boldsymbol{c})$ (31). A pesar de que procediendo de esta forma nos expondríamos a una posible degradación del rendimiento de clasificación, este método nos va a proporcionar una clasificación de subconjuntos de muestras que va a concluir más rápidamente que si las eliminásemos individualmente.

Nosotros hemos desarrollado una implementación del SVM-RFE utilizando la librería *libsvm* de *Matlab* (APÉNDICE C: LIBSVM) para entrenar el clasificador. Además hemos decidido que, dado que contamos con miles de dimensiones en nuestros datos, éstas serán eliminadas de diez en diez, con el fin de que nuestro algoritmo sea ejecutado más rápidamente. Debajo de estas líneas se explica paso a paso el código implementado para este algoritmo.

En primer lugar, volver a mencionar que utilizaremos las muestras y las etiquetas almacenadas en las bases de datos que más adelante describiremos (*Carcinom, Glioma* y *Lung*), el valor de coste óptimo obtenido para la SVM multiclase y el algoritmo *leave-one-out* para llevar a cabo la clasificación.

Para el método RFE utilizaremos nuevas variables. En primer lugar necesitamos una para almacenar el conjunto total de dimensiones de los datos de entrada (*total_posiciones*), cuya longitud iremos reduciendo a medida que avancemos en el desarrollo del algoritmo, en función de las dimensiones que escojamos como menos relevantes, que serán eliminadas.

Además, con el objetivo de extraer la mayor cantidad de información posible, en esta implementación del RFE calculamos la matriz de confusión. Esto nos va a permitir comprobar si los errores se reparten uniformemente entre todas las clases o si, por el contrario, hay clases correspondientes a determinadas patologías en las que el número de fallos es mayor. Esta matriz de confusión no es más que una matriz cuadrada en la que el número de filas es igual al número de clases. Las filas se corresponden con las clases verdaderas, y las columnas con las clases predichas por el clasificador. Por este motivo necesitaremos otra variable auxiliar para almacenar las etiquetas que se le vayan asignando a las muestras en cada iteración. El número de filas de esta variable corresponderá a las diferentes observaciones y las columnas representan las dimensiones de cada una de las muestras.

Antes de comenzar con el desarrollo de nuestro algoritmo de extracción de muestras (*RFE*), consideramos importante anotar que la implementación del algoritmo *leave-one-out* va a ser muy similar a cómo hemos estado procediendo hasta ahora: del conjunto total de muestras extraemos una muestra para realizar el test y el resto lo dejamos para llevar a cabo el entrenamiento.

Seguidamente introducimos el bucle del *RFE*, cuyo número de iteraciones dependerá de las dimensiones de los datos a clasificar. Esto es así porque el criterio que establecemos para detener el bucle es que la longitud de *total_posiciones* sea mayor a 10 y aunque inicialmente este valor es igual al número de variables que posee cada muestra, a medida que avancemos en el desarrollo iremos extrayendo de diez en diez las dimensiones que consideremos menos significativas en función del valor de los pesos w_i de la SVM. Por este motivo es necesario apuntar que el vector *total_posiciones* debe ser inicializado a su valor original una vez finalizado el *RFE* para poder procesar correctamente una nueva iteración del *Leave-One-Out*.

Puesto que el número de dimensiones de los datos de entrada irán variando en cada iteración, necesitaremos volver a calcular la matriz de kernels cada vez, de ahí que cuando hacemos el producto de las matrices de muestras *M*, sólo tenemos que seleccionar las columnas indicadas en *total_posiciones*.

Nótese que el entrenamiento del clasificador se realiza de manera análoga utilizando la función *svmtrain*, y obteniendo las mismas salidas que en el caso de las SVM multiclase. Sin embargo, dado que de momento nuestro objetivo es únicamente determinar cuáles son los genes con mayor relevancia dentro de la secuencia, no vamos a utilizar la función *svmpredict* ni a almacenar las etiquetas asignadas a cada muestra.

Posteriormente incluímos el código necesario para llevar a cabo el método RFE. De la función *svmtrain* obtenemos los vectores de soporte propios de la SVM, que *Matlab* denomina *sv_indices*. Estos índices son los que determinan el número de máquinas de vectores de soporte que utilizará nuestro algoritmo. Asimismo, nos permitirán extraer qué elementos de la matriz de datos de entrada *M*, serán utilizados cómo valores x_i en la fórmula $\boldsymbol{w = \sum_k \alpha_k y_k x_k}$ 29. Por otra parte, los coeficientes α_i los obtendremos también de la salida de la SVM, concretamente de *sv_coef*. A partir de estos dos parámetros podremos calcular los pesos w_i, para finalmente obtener como resultado $w = \sum_k \alpha_k x_k$, que era lo que buscábamos desde un principio.

Dado que el nuestro es un problema multiclase, α será una matriz de tantas filas como clases distintas existan en la base de datos y tantas columnas como máquinas de vectores soporte nos devuelva el *svmtrain*, por lo que al multiplicarla por los datos de entrada nos devolverá una matriz de vectores de pesos, w. Por ello, para poder establecer un ranking y descartar después las dimensiones menos relevantes, utilizamos la función *sum* con los valores absolutos de los pesos y así obtenemos una *w* final más adelante ordenaremos.

Para extraer las dimensiones menos relevantes de nuestros datos de entrada, lo que hacemos es, una vez ordenados los pesos de menor a mayor, vamos concatenando las variables menos discriminatorias de diez en diez para cada iteración del *RFE*, con el objetivo de determinar al final del algoritmo la relevancia real de cada una de ellas.

En el momento en el que ha concluido el *RFE*, añadimos al vector en el que almacenamos anteriormente las variables menos decisivas aquellas a las que el

algoritmo ha considerado como más relevantes y que, por tanto, no han sido eliminadas. Puesto que este algoritmo se llevará a cabo tantas veces como muestras tengamos (una por cada iteración del bucle *leave-one-out*) necesitamos almacenar en qué orden son eliminadas las muestras en cada caso. Además, con el fin de conocer la relevancia real de los genes en la clasificación llamamos a la función *sort* para que nos devuelva el momento en el que fue eliminada cada variable. Esta información la almacenaremos en una matriz de las mismas dimensiones que la matriz de muestras *M* (cada fila representa a una observación y cada columna a una dimensión de la secuencia genética).

Una vez que conocemos cuáles son las variables que más influyen en el proceso de clasificación de nuestras muestras, vamos a repetir el proceso utilizando sólo las 200 más relevantes. Esto nos permitirá comparar después las ordenaciones de los genes realizadas por los dos algoritmos desarrollados. Además, más adelante analizaremos cómo la selección de conjuntos de muestras de distintos tamaños influye en los resultados de la clasificación.

Posteriormente elaboramos un promedio con el orden en el que fueron extraídas las dimensiones en cada iteración del *leave-one-out*. De esta forma podremos determinar qué posiciones dentro de la secuencia han resultado en general más discriminatorias y nos aportarían más información a la hora de establecer un diagnóstico.

Como ya explicamos al principio, en esta implementación del RFE vamos incluir el cálculo de la matriz de confusión para así obtener una visión de cómo se reparten los errores a lo largo de la ejecución. Por este motivo creamos un primer bucle *for* y vamos almacenando todas las etiquetas de *L* que representarán a las clases verdaderas (y repetimos el bucle tantas veces como clases distintas tenemos). A continuación, como lo que queremos es rellenar una matriz cuadrada introducimos otro bucle que represente las clases predichas por el clasificador. Con este fin vamos almacenando en cada columna de la matriz de confusión el número de etiquetas totales de la matriz de etiquetas (en las filas de la clase original y con el número de dimensiones que nos proporcione una mayor tasa de acierto) que sean iguales a la iteración en la que nos encontremos, designada esta vez por *n*.

Al finalizar este doble bucle *for*, obtendremos la matriz correspondiente y concluirá la ejecución del algoritmo RFE utilizando las máquinas de vectores de soporte. De este algoritmo podremos extraer varios datos importantes (matrices de confusión, orden promedio de eliminación de las variables, tasas de acierto…) que describiremos al final del estudio.

3.3 RFE ROBUSTO

Hasta ahora, todo lo que hemos estado viendo son algoritmos de clasificación que ya habían sido probados para distintas clases de conjuntos de datos y que sabemos de antemano para qué casos resulta óptima su utilización o para cuáles sería más conveniente el uso de otro tipo de procedimientos.

Dado que el objetivo de nuestro trabajo es ser capaces de establecer un criterio que nos permita determinar qué genes son más relevantes dentro de una secuencia con miles de unidades de la manera más eficiente posible, hemos decidido introducir una modificación en el algoritmo RFE, de tal forma que resulte mucho más robusto estadísticamente. Con esta variación lo que buscamos es estar casi completamente seguros de la relevancia real de cada una de las variables que hemos seleccionado, teniendo en cuenta que nuestro propósito no es optimizar el acierto de clasificación sino encontrar el mejor compromiso entre una tasa de acierto elevada y la elección de las variables que resultan más determinantes en la clasificación.

Para llevar a cabo este RFE robusto hemos decidido emplear una técnica de re-muestreo conocida como *bootstrap* [9], que permite mejorar la precisión en la clasificación mediante la incorporación de aleatoriedad en la construcción de cada clasificador individual. Para ello vamos a introducir un bucle en el que se repita la ejecución del algoritmo *recursive feature elimination* durante 1000 iteraciones. Es importante remarcar que el hecho de haber escogido ese número de iteraciones es la positividad de los resultados que obtuvimos al utilizar este valor, aunque de la misma forma podríamos haber elegido otro e ir probando hasta alcanzar el rendimiento adecuado. Para cada iteración de este bucle lo que haremos será elegir un subconjunto de observaciones al azar. Vamos a construir el conjunto de datos escogiendo de manera aleatoria el 50% de los datos disponibles únicamente. De cada RFE iremos almacenando el orden en el que se eliminan las dimensiones para cada ejecución y así, una vez finalizado el algoritmo, podremos determinar cuál es la posición promedio de cada variable. Este sistema nos proporcionará así una idea de la relevancia de cada una de ellas.

Al igual que hicimos en capítulos anteriores, vamos a incluir a continuación una descripción del código que desarrollamos para llevar a cabo este algoritmo, que incluye ciertas diferencias con respecto al RFE original.

En la primera fase de la implementación apenas aparecen modificaciones del código anterior. Cargamos la base de datos que vamos a utilizar, seleccionamos el parámetro de coste óptimo para llevar a cabo la clasificación, guardamos el número de *observaciones* y de *dimensiones* que tenemos en nuestra matriz de muestras (*M*) e inicializamos el

vector *total_posiciones* que utilizábamos para el RFE y la matriz de confusión para conocer el reparto de los errores.

Inmediatamente después de la declaración e inicialización de las variables incluimos las primeras sentencias del bucle *leave-one-out* que, como ya se ha explicado anteriormente, deja una de las muestras del conjunto para utilizarla como test y el resto como entrenamiento.

A partir de este punto es donde aparece la parte novedosa del algoritmo. Creamos un bucle *for* que se repita durante mil iteraciones. En cada iteración asignamos a un vector valores al azar que se encuentren dentro de los índices utilizados para entrenamiento de la matriz de observaciones. Sin embargo, de estos valores aleatorios nos quedamos con únicamente la mitad y a partir de estos datos construimos las nuevas matrices de observaciones y etiquetas.

Una vez que tenemos disponibles los datos a utilizar, procedemos de forma similar a como habíamos estado haciendo hasta ahora, guardamos el número de observaciones y las dimensiones de la nueva matriz de muestras e inicializamos los nuevos índices de entrenamiento con los que contamos. Antes de continuar con el resto del código es importante volver a mencionar que en este punto resulta necesario inicializar el vector *total_posiciones*, puesto que tras la ejecución de todas las iteraciones del bucle RFE original las dimensiones de este vector serán las que hayan quedado sin extraer al finalizar el algoritmo.

Dentro del bucle utilizado para hacer el algoritmo más robusto incluímos el RFE estándar descrito en el apartado anterior. Lo que hemos hecho ha sido introducir una nueva matriz en la que almacenaremos, para cada iteración distinta que realicemos de cada RFE con sus propios datos, el orden de extracción de las dimensiones. Esto nos facilitará posteriormente llevar a cabo un promedio sobre cuándo ha sido eliminada cada variable, lo que nos proporcionará una idea acerca de la relevancia real de cada una de ellas.

Una vez que han finalizado las mil iteraciones es el momento de realizar la media que nos permita saber en qué posición se eliminan las variables de la secuencia. Seguidamente realizamos una ordenación que nos permita averiguar cómo de concluyente resulta cada dimensión para el proceso de clasificación y predicción posterior. Puesto que estamos implementando un algoritmo *leave-one-out* vamos a tener tantas filas de variables ordenadas como observaciones aparezcan en la matriz de muestras.

Ahora que ya tenemos un vector con los genes por orden de relevancia podemos volver a calcular la matriz de kernels para volver a aplicar las funciones *svmtrain* y *svmpredict* y llevar a cabo la clasificación final de las muestras. Con el objetivo de establecer una posterior comparación con el *RFE* normal, aquí también utilizamos sólo las 200 dimensiones más relevantes.

Finalmente, cuando ya hemos terminado todas las iteraciones del *LOO*, realizamos un promedio global de todas las variables en orden de relevancia con el objetivo de no dejar excluida ninguna de las muestras que se utilizan como test. Además, al igual que hicimos con el *RFE* estándar, procedemos al cálculo de la matriz de confusión para conocer cómo se reparten los errores entre las diferentes clases.

CAPÍTULO 4: EXPERIMENTOS

4.1 CONJUNTOS DE DATOS

4.2 DESCRIPCIÓN DE LOS RESULTADOS

4.3 CONCLUSIONES SOBRE LOS RESULTADOS

4.1 CONJUNTOS DE DATOS

Para la realización de este estudio nos hemos basado en tres bases de datos de secuencias genéticas que están disponibles públicamente [3]. En estos conjuntos de datos se encuentra almacenada parte de la información genética de pacientes que han desarrollado diferentes tipos de cáncer. En cada uno de ellos tendremos dos matrices distintas, una llamada *L* que contendrá las etiquetas, y otra llamada *M* que reunirá el conjunto de todas las muestras.

A continuación aparece detallada la información de cada base de datos por separado:

- Carcinom: este dataset [8] contiene en total 174 muestras pertenecientes a 11 clases distintas de cáncer. Estas clases son: próstata, vejiga/uréter, mama, colorrectal, gastroesofágico, riñón, hígado, ovario, páncreas, adenocarcinomas de pulmón y carcinoma de células escamosas, que cuentan con 26, 8, 26, 23, 12, 11, 7, 27, 6, 14 y 14 muestras respectivamente. En este conjunto cada muestra contiene, a su vez, 9182 genes. Puesto que en él se ha incluido un amplio espectro de modalidades de células cancerosas no es de extrañar que para realizar la clasificación de las muestras se haya tenido en cuenta casi el 50% del genoma humano (el total corresponde a unos 20.000 genes).

- Lung: ahora nos enfrentamos a un conjunto de 203 muestras repartidas en cinco clases [7], entre las que se encuentran cuatro tipos diferentes de cáncer en la zona pulmonar (adenocarcinomas, carcinomas escamosos de células pulmonares, carcinoides pulmonares, carcinomas de pulmón de células pequeñas). La última clase del dataset, por el contrario, corresponde a observaciones pertenecientes a pacientes con pulmones sanos. Para cada una de las muestras de la base de datos existen 3312 genes. Resulta lógico que se haya seleccionado un fragmento tan pequeño de material genético dado que únicamente necesitamos información relativa a una zona muy concreta.

- Glioma: el dataset Glioma [6] contiene un total de 50 muestras de 4 clases relativas a cánceres desarrollados en el sistema nervioso, a saber, gliobastomas cancerosos, gliobastomas no cancerosos, oligodendrogliomas cancerosos y oligodendrogliomas no cancerosos, que presentan 14, 14, 7 y 15 muestras

respectivamente. Las secuencias de expresión genética de cada sujeto contienen 4433 genes. En esta ocasión, como las células del sistema nervioso son más numerosas y se hayan más repartidas por el organismo que en el caso del dataset *Lung* necesitamos un espectro mayor de información genética.

4.2 DESCRIPCIÓN DE LOS RESULTADOS

Una vez que ha concluido la ejecución de los algoritmos implementados vamos a proceder a comentar los resultados obtenidos. De momento vamos a realizar la explicación por separado de los datos del *RFE* y el *RFE* robusto para una clasificación en la que hemos empleado solamente 200 dimensiones. Más adelante analizaremos las prestaciones de uno y otro para cada una de las bases de datos disponibles y mostraremos cómo varían los resultados para cada algoritmo en función del número de variables que utilicemos en la clasificación final.

RFE

Carcinom

- ***Matriz de confusión***

	CLASES PREDICHAS										
CLASES VERDADERAS	26	0	0	0	0	0	0	0	0	0	0
	0	4	0	1	0	0	0	1	0	1	1
	1	0	22	0	0	0	1	0	0	2	0
	0	1	0	21	0	0	0	0	1	0	0
	0	1	1	2	7	0	0	0	0	1	0
	0	0	0	1	0	10	0	0	0	0	0
	0	0	2	0	0	0	4	1	0	0	0
	0	0	0	0	0	0	0	27	0	0	0
	0	1	0	1	1	0	0	0	3	0	0
	0	1	1	0	0	0	0	0	2	9	1
	0	0	1	0	0	0	0	0	0	1	12

Tabla 4.1. Distribución de muestras para la base de datos *Carcinom* con el algoritmo RFE.

Entre las once clases con las que cuenta este dataset, la mayoría de las muestras se reparten entre la primera (cáncer de próstata), la tercera (cáncer de mama), la cuarta (cáncer colorrectal) y la octava (cáncer de ovarios). Sin embargo, no es en ellas donde aparece una cantidad más elevada de errores, lo que es totalmente normal ya que al tener más datos la SVM ha tenido más ejemplos para aprender. Las clases que cuentan con una menor tasa de acierto son la segunda (cáncer de vejiga/uréter), la quinta (cáncer

gastroesofágico), la séptima (cáncer de hígado) y la novena (cáncer de pancreas) que fallan con el 50% de muestras que clasifican aproximadamente. En la clase número dos los errores se reparten equitativamente entre las clases cuatro, ocho, diez (adenocarcinomas de pulmón) y once (carcinoma de células escamosas); para la número cinco, los errores se concentran sobre todo en la clase cuatro; en la clase siete la confusión aparece especialmente con la clase número tres, pero también se producen errores con la ocho; y por último, en la novena clase las muestras mal clasificadas también aparecen repartidas, en este caso, entre las clases dos, cuatro y cinco.

- ***Ranking de las variables en orden creciente de relevancia***

Puesto que en esta base de datos las secuencias genéticas de cada muestra cuentan con 9182 dimensiones, no vamos a discutir sobre la posición de todas ellas. No obstante, hemos decidido mostrar las que hemos considerado más significativas a la hora de establecer una posterior comparación.

Dimensiones menos relevantes

926	1986	7435	4093	6468	1937	2507	80	5297	3518	8610	3229	8890	4137

Dimensiones más relevantes

8246	8252	8453	8455	8636	8640	8656	8672	8740	8930	9005	9102	9116	9164

Glioma

- ***Matriz de confusión***

	CLASES PREDICHAS			
CLASES VERDADERAS	10	3	1	0
	3	4	0	0
	0	0	10	4
	0	0	3	12

Tabla 4.2. Distribución de muestras para la base de datos *Glioma* con el algoritmo RFE.

En esta base de datos las muestras se hallan más equitativamente repartidas, al igual que la tasa de error. En la primera clase (gliobastomas cancerosos) el porcentaje de fallo es del 28% y aparece sobre todo con la segunda clase (gliobastomas no cancerosos), aunque también con la tercera (oligodendrogliomas cancerosos) en alguna ocasión. La clase en la que se producen menos aciertos es la segunda. En este caso la confusión es entre pacientes con gliobastomas no cancerosos y aquellos con gliobastomas cancerosos, lo que resulta un grave problema a la hora de establecer un dictamen clínico. En la tercera clase tenemos un caso similar al anterior, aparecen errores entre

los pacientes con células cancerosas y aquellos en los que no se ha manifestado dicha patología, con un porcentaje de error algo menor al que se mostraba con los gliobastomas. En último lugar se encuentran los sujetos con oligodendrogliomas no cancerosos, que son aquellos más fáciles de detectar.

- ***Ranking de las variables en orden creciente de relevancia***

Al igual que en el dataset anterior, ahora vamos a introducir dos tablas en las que se muestren, por un lado, las variables menos relevantes, y por otro, las más relevantes.

Dimensiones menos relevantes

2168	2381	2551	570	4388	3889	2485	2192	2774	2907	2646	1023	1022	1904

Dimensiones más relevantes

4334	4367	4381	4387	4393	4394	4414	4420	4422	4423	4424	4425	4426	4434

Lung

- ***Matriz confusión***

	CLASES PREDICHAS				
CLASES VERDADERAS	135	1	2	0	1
	3	14	0	0	0
	5	0	16	0	0
	0	0	0	20	0
	0	0	1	0	5

Tabla 4.3. Distribución de muestras para la base de datos *Lung* con el algoritmo RFE.

En último lugar vamos a comentar la base de datos *Lung*. En la primera clase (adenocarcinomas), que es la que cuenta con un mayor número de muestras, solamente se produce porcentajes de error alrededor del 1% con la segunda (carcinomas escamosos de células pulmonares), la tercera (carcinoides pulmonares) y la quinta clase (pulmones normales). En las clases dos y tres también se producen errores con la primera clase, aunque en estos casos con un porcentaje algo mayor (del 17% y el 24%, respectivamente). La clase cuatro (carcinomas de pulmón de células pequeñas), a pesar de no ser la que cuenta con un número más elevado de muestras, nos proporciona una tasa de acierto del 100%. Para terminar, la quinta clase confunde sus muestras únicamente con la clase tres.

- ***Ranking de las variables en orden creciente de relevancia***

Procedemos ahora a mostrar cómo han sido ordenadas algunas de las variables de la secuencia con el objetivo de determinar su relevancia en la clasificación.

Dimensiones menos relevantes

657	151	3089	373	720	3147	342	2153	418	332	641	419	934	1641

Dimensiones más relevantes

3151	3153	3156	3218	3239	3244	3247	3248	3252	3273	3275	3282	3292	3307

Conclusiones RFE

A pesar de que en este punto todavía no somos capaces de establecer una conclusión definitiva sobre la clasificación de nuestras muestras con el algoritmo utilizado, ya que lo idóneo es compararlo con el *RFE* robusto, realizaremos un breve análisis de los resultados obtenidos.

En primer lugar vamos a comenzar comentando los resultados obtenidos para la matriz de confusión. Una vez finalizado el cálculo hay que resaltar que no es de extrañar el hecho de que en general se haya producido un menor porcentaje de acierto en las clases en las que teníamos menos muestras. Si de un total de unos cien elementos apenas diez pertenecen a una clase en concreto, es normal que se produzcan más errores que en otra de la que tenemos casi treinta muestras, ya que el número de ocasiones que va a poder entrenar con las muestras minoritarias va a ser mucho más reducido. Por tanto, a la hora de realizar el test resulta mucho más probable que se produzcan confusiones en las clases con menos observaciones. Además si nos fijamos, estos errores aparecer con más frecuencia con clases con las que teníamos más muestras para clasificar. Esto es así porque el algoritmo "aprende" cuáles son las variables más relevantes en función de una serie de observaciones utilizadas para entrenar. Si contamos con más observaciones de un determinado tipo al final las variables escogidas como más relevantes serán las que en promedio resulten más relevantes para la clasificación de esas muestras en concreto, lo que no significa que también sean determinantes para datos de otro tipo.

Aun así es importante mencionar la elevada tasa de acierto que hemos obtenido para cada uno de los conjuntos utilizados. A pesar de que no contábamos con un número demasiado elevado de observaciones para desarrollar un proceso de entrenamiento más extenso, el porcentaje de error obtenido es en promedio del 21%, siendo mayor en el primero y el segundo, y menor en el tercero. Una de las causas de haber obtenido un menor porcentaje de acierto en el primer y el segundo dataset puede ser la gran variedad

de clases a las que pueden pertenecer las muestras. Los datos de *Carcinom* o *Glioma*, al contrario que en *Lung* que son específicos de zonas o sistemas concretos del organismo, están referidos a tipos de cáncer que afectan a un mayor espectro de elementos en la anatomía de un individuo. Esto hace que el fragmento de secuencia genética que se haya seleccionado, además de ser mayor, deba incluir genes relacionados con un espectro más amplio de aspectos biológicos.

El hecho de que la agrupación las bases de datos en función de determinadas partes del organismo resulta muy positivo en la clasificación se observa sobre todo en el dataset *Lung*. Al ser todas las clases relativas a cánceres en el pulmón, la longitud del array es mucho menor (únicamente 3312 elementos), y la mayor parte de los genes quedan referidos a aspectos relacionados con este tipo de células.

Por otra parte es necesario hacer referencia al orden de eliminación de las variables. En el caso de las más relevantes suelen aparecer posiciones cercanas, generalmente consecutivas. Este hecho choca con las enormes diferencias que existen entre aquellas menos relevantes, para las que en ningún caso observamos dimensiones contiguas. Las razones por las que esto sucede son varias. En el caso de las variables más relevantes, el algoritmo escoge casi siempre para todas las iteraciones del *leave-one-out* a un mismo conjunto como las más decisivas y, puesto que siempre van agrupadas de diez en diez, resulta lógico que aparezcan posiciones que se siguen. En el caso de las menos relevantes, cabe destacar que la correlación existente entre determinadas variables hace que el *RFE* no sea capaz de distinguir entre una y otra, ya que ambas le van a resultar igualmente irrelevantes a la hora de clasificar. Por este motivo, habrá iteraciones en las que escogerá unas, e iteraciones en las que escoja otras totalmente diferentes. El resultado será un promedio final en el que aparecerán datos muy dispares respecto a estas variables menos concluyentes.

RFE ROBUSTO

Vamos a proceder ahora a analizar los datos extraídos de la ejecución de nuestro algoritmo basado en múltiples ejecuciones del *RFE* con diferentes observaciones en cada una.

Carcinom

- ***Matriz confusión***

	CLASES PREDICHAS										
CLASES	26	0	0	0	0	0	0	0	0	0	0
	0	6	1	0	0	0	0	0	0	0	1
	1	0	25	0	0	0	0	0	0	1	0
	0	0	0	23	0	0	0	0	0	0	0
	0	0	0	0	11	0	0	0	1	0	0

VERDADERAS	0	0	0	0	0	11	0	0	0	0	0
	0	0	1	0	0	0	6	0	0	0	0
	0	0	0	0	0	0	0	27	0	0	0
	0	0	0	1	0	0	0	0	5	0	0
	0	0	1	0	0	0	0	0	0	13	1
	0	0	0	0	0	0	0	0	0	0	14

Tabla 4.4. Distribución de muestras para la base de datos *Carcinom* con el algoritmo RFE robusto.

Dado que los datos a procesar son los mismos, las clases que cuentan con mayor cantidad de muestras siguen siendo la primera, la tercera, la cuarta y la octava. No obstante, ahora los errores han disminuido con respecto a la implementación del *RFE* convencional.

Tal y como se puede observar en la imagen de arriba, se ha incrementado notablemente el número de clases en las que se obtiene una tasa de aciertos del 100% (ahora son la primera, la cuarta, la sexta, la octava y la undécima). No obstante, en aquellas en las que siguen apareciendo muestras incorrectamente clasificadas el porcentaje de éstas es mucho menor que con el anterior algoritmo. Si ordenamos las clases en función del número de fallos que acumulan podemos concluir que aquellas que abarcan una mayor cantidad de muestras incorrectamente clasificadas son la segunda (un 25%), la novena (16%) y la séptima (14%), seguidas por la quinta, la décima y la tercera, estas últimas con un porcentaje de error inferior al 10%.

- ***Ranking de las variables en orden creciente de relevancia***

De la misma forma que comentamos anteriormente con el *RFE* normal, hemos hecho una selección de las variables que considerábamos más importantes a la hora de sacar conclusiones.

Dimensiones menos relevantes

1878	4045	4918	4922	4666	4131	4143	3354	3933	3713	3546	3606	3808	3548

Dimensiones más relevantes

8244	8251	8425	8455	8636	8640	8656	8672	8740	8930	9005	9102	9116	9164

Glioma

- ***Matriz confusión***

	CLASES PREDICHAS			
CLASES VERDADERAS	13	1	0	0
	1	6	0	0
	0	0	11	3
	0	0	2	13

Tabla 4.5. Distribución de muestras para la base de datos *Glioma* con el algoritmo RFE robusto.

En este dataset también observamos que el número de muestras incorrectamente clasificadas no es tan elevado como ocurría con *RFE* estándar. Aunque los errores son menores, aparecen repartidos de manera similar a como sucedía con el anterior algoritmo. Salvo en la primera clase, que ahora confunde un 7% de sus muestras con la segunda únicamente, el resto sigue teniendo errores con las mismas clases que en el *RFE* normal, aunque en un porcentaje menor.

- ***Ranking de las variables en orden creciente de relevancia***

Dimensiones menos relevantes

50	1717	451	561	2056	1565	2202	258	2507	3889	2551	758	2907	2032

Dimensiones más relevantes

4334	4298	4381	4367	4387	4393	4394	4420	4422	4423	4424	4425	4426	4434

Lung

- ***Matriz confusión***

	CLASES PREDICHAS				
CLASES VERDADERAS	138	0	1	0	0
	1	16	0	0	0
	5	0	16	0	0
	0	0	0	20	0
	0	0	0	1	5

Tabla 4.6. Distribución de muestras para la base de datos *Lung* con el algoritmo RFE robusto.

Para esta última base de datos podemos apreciar cómo se han reducido las tasas de error especialmente para la primera y la segunda clase. No ocurre lo mismo, sin embargo, con las clases restantes. Los aciertos en la tercera y la cuarta se mantienen igual que en el *RFE*. En la clase número cinco, aunque se producen los mismos fallos, en este caso se producen con la clase cuatro en lugar de con la tres como ocurría anteriormente.

- ***Ranking de las variables en orden creciente de relevancia***

Dimensiones menos relevantes

2153	404	3147	332	720	2170	418	2929	422	419	373	151	521	589

Dimensiones más relevantes

3096	3131	3151	3153	3156	3218	3239	3247	3248	3273	3275	3282	3292	3307

Conclusiones RFE robusto

Ahora que hemos extraído y analizado los resultados para todas las bases de datos de este *RFE* robusto vamos a proceder realizar algunos comentarios acerca de los mismos. Al igual que hicimos con el anterior *RFE*, hablaremos tanto de la tasa de aciertos como de la ordenación de las variables, ya que lo que buscamos es alcanzar un compromiso entre ambas.

Respecto a la tasa de aciertos, podemos concluir que la implementación del *RFE* robusto utilizando la técnica de *bootstrap* ha sido exitosa. Hemos obtenido una mayor tasa de acierto que queda reflejada en la diagonal de las matrices de confusión que hemos ido calculando, donde podemos observar que el número de muestras clasificadas correctamente es más elevado que en el caso anterior.

Si nos fijamos ahora en la nueva ordenación que el *RFE* robusto hace de los genes de la secuencia observamos varios aspectos a tener en cuenta. Por un lado apreciamos que las variables que el algoritmo considera como más relevantes son las mismas que en el caso anterior. No ocurre lo mismo sin embargo para el resto de dimensiones de la secuencia. La fuerte correlación entre ciertos genes y los sucesivos promedios que hacemos de los distintos ordenes en los que son extraídos en cada iteración (al acabar el *RFE* robusto, el *leave-one-out*…) hace que sea más difícil la aparición de elementos iguales entre los vectores del ranking de variables en función del momento en el que son extraídas de los dos algoritmos. Si realizamos una comparación detallada de ambos vectores de relevancia de los genes podemos apreciar que es para la base de datos *Carcinom* para la que aparece un mayor número de variables en la misma posición (las primeras sesenta posiciones son prácticamente las mismas). No es así para *Lung* o *Glioma*. En estos casos coinciden un número menor de posiciones (alrededor de las diez primeras). No obstante es importante mencionar que aquellos genes que resultan más determinantes a la hora de realizar la clasificación no varían en ninguno de los algoritmos, lo que nos da una idea acerca del correcto funcionamiento de ambos.

Por todas estas razones podemos concluir que, a pesar de que el *RFE* robusto conlleve un mayor coste computacional y, por tanto, un mayor tiempo de ejecución, los resultados obtenidos son bastante más favorables si los comparamos con los del *RFE*. Con él obtenemos una tasa de error bastante más reducida, que para la finalidad de nuestro estudio resulta imprescindible ya que es la única manera que tenemos para medir la relevancia. Además, puesto que lo que estamos analizando son bases de datos en las que las muestras corresponden a pacientes en los que, en la mayoría de los casos, se ha desarrollado algún tipo de cáncer parece evidente que alcanzar un nivel elevado de precisión en la clasificación sea algo prioritario.

Pero no solo eso. Con el *RFE* fuimos capaces de extraer de secuencias genéticas de miles de elementos aquellos que solamente aportaban ruido y no nos permitían llegar a ninguna conclusión, seleccionando así conjuntos cuya influencia era decisiva en la clasificación. Este ordenamiento de los genes quedó ratificado poco después cuando implementamos la versión robusta del *RFE*. El nuevo algoritmo implementado por nosotros nos ha permitido no sólo asegurarnos de la verdadera relevancia de las dimensiones en las muestras sino también extraer determinados elementos que antes se habían pasado por alto y que, sin embargo, sí que resultan relevantes a la hora de establecer un diagnóstico.

4.3 CONCLUSIONES SOBRE LOS RESULTADOS

Una vez que hemos obtenido y comentado los resultados para ambos algoritmos vamos a llevar a cabo un análisis comparativo que nos permita analizar de manera más detallada las diferencias existentes entre las dos implementaciones.

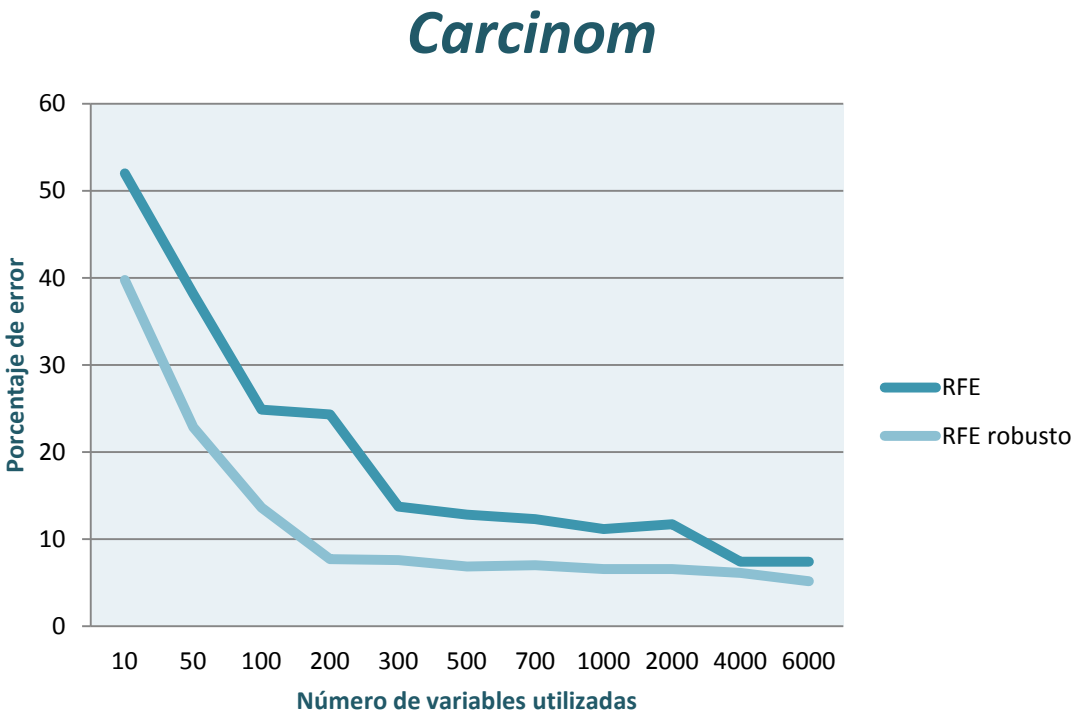
A continuación hemos introducido una tabla a modo de resumen en la que podremos visualizar los promedios de error que se han obtenido en las tres bases de datos para los distintos algoritmos.

	RFE	**RFE ROBUSTO**
Carcinom	24.32%	7.69%
Glioma	29.9%	14.04%
Lung	12.2%	6.86%

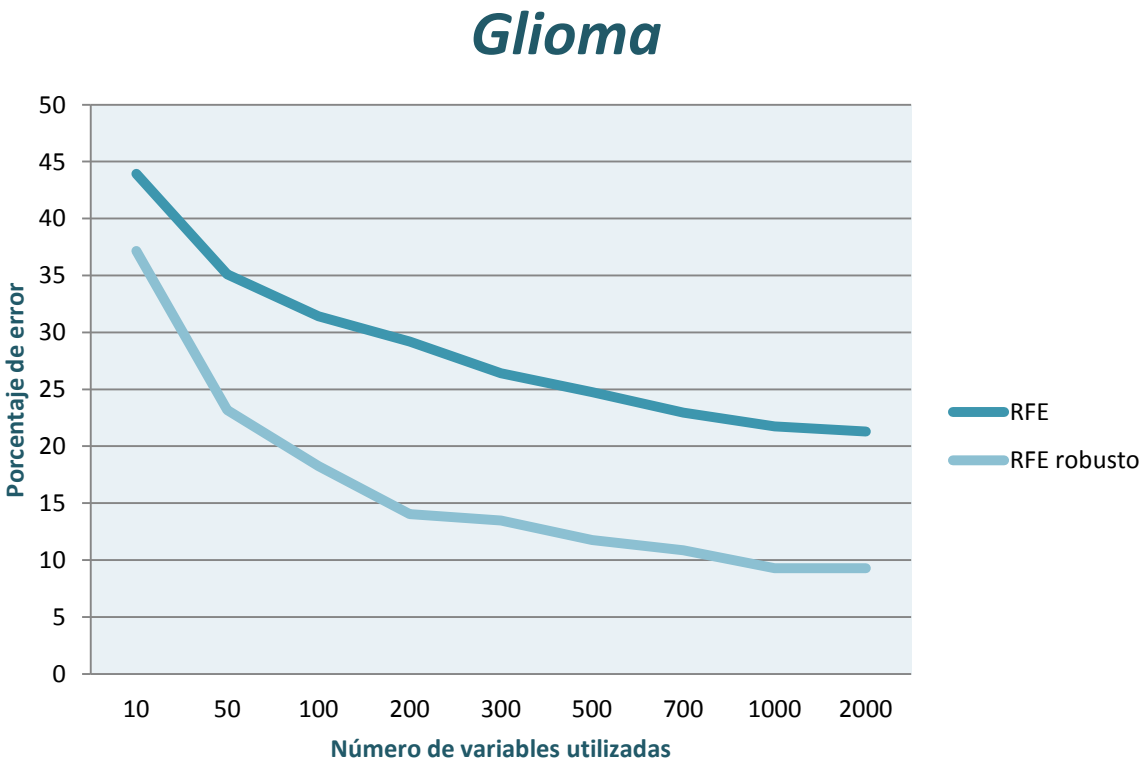
En esta tabla se observa que, a pesar de haber necesitado un mayor tiempo de computación para ejecutar el *RFE* robusto, esto no nos supone un error ya que nos permite alcanzar una menor tasa de error, que llega a reducirse hasta dieciséis puntos porcentuales en el caso del dataset *Carcinom*. Los porcentajes de fallo más elevados corresponden a *Glioma*, lo que no es de extrañar ya que era la base de datos que contaba con un menor número de muestras para realizar el proceso de entrenamiento en la clasificación. Es importante volver a mencionar en este punto que el interés por obtener una elevada tasa de acierto reside en que es nuestra manera de medir si hemos fracasado o no en la selección de los genes más relevantes.

Podemos confirmar entonces que la introducción de la técnica de *bootstrap* en el *RFE* nos da lugar a unos resultados mucho más precisos y, por tanto, más fiables. A partir de estos resultados nos podremos hacer una idea sobre la verdadera relevancia de cada uno de los genes que componían las secuencias con las que estábamos trabajando.

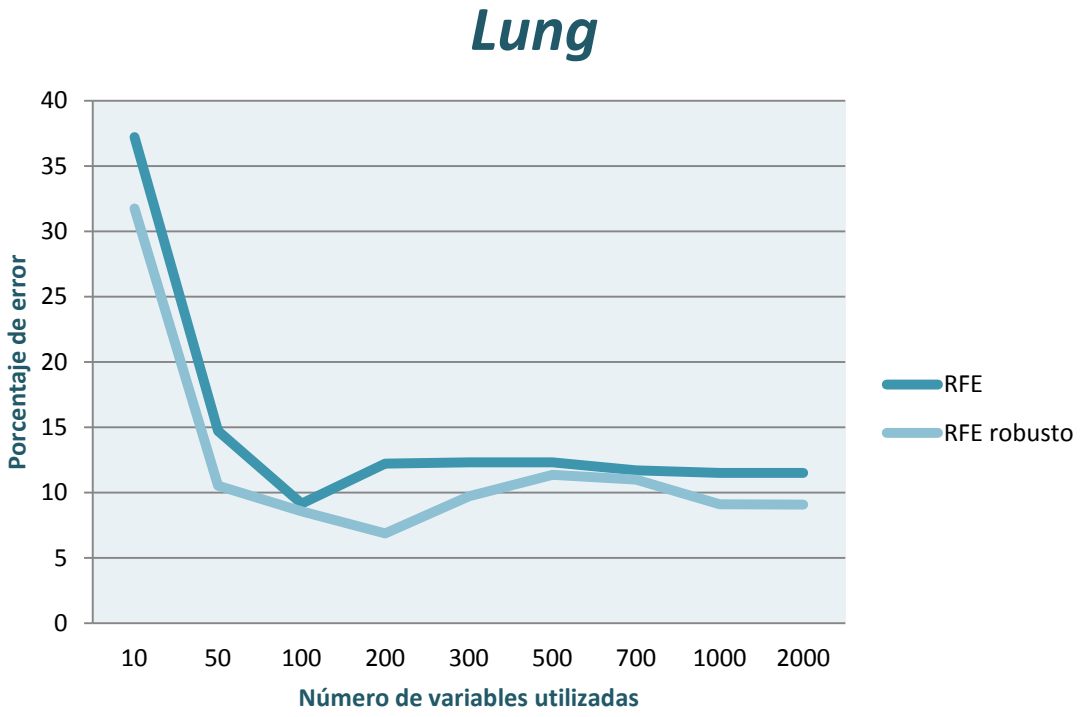
Puesto que todos estos resultados habían sido obtenidos a partir de un proceso de clasificación en el que sólo empleamos los 200 genes más relevantes de la secuencia, hemos decidido comprobar cómo variaría el porcentaje de error si modificásemos el número de variables empleadas en la última fase de la clasificación para cada una de las bases de datos disponibles. Para ello vamos a repetir la ejecución del algoritmo y a realizar la clasificación final con conjuntos de genes de diferentes tamaños. De esta forma podremos observar las tasas de error que se obtienen en función de la cantidad de variables que sean escogidas como las más concluyentes.



Gráfica 4. 1. Porcentaje de error obtenido en la matriz de confusión de Carcinom para secuencias de distintos tamaños.



Gráfica 4. 2 Porcentaje de error obtenido en la matriz de confusión de Glioma para secuencias de distintos tamaños.



Gráfica 4. 3 Porcentaje de error obtenido en la matriz de confusión de Lung para secuencias de distintos tamaños.

A partir de estas gráficas se puede extraer que, en general, el empleo de un mayor número de variables para realizar la clasificación nos permite reducir la tasa de error. Esta disminución de la tasa de error es bastante significativa si pasamos de clasificar con unas diez o cincuenta dimensiones a hacerlo con cien o doscientas. No obstante, se puede apreciar como a partir de las trescientas dimensiones los resultados obtenidos no mejoran significativamente. Esto es así porque a pesar de que un gran número de variables pueden llegar a aportarnos algún tipo de información, son especialmente las más relevantes las que nos ayudan a clasificar correctamente. Este hecho nos permite además volver a ratificar que hemos acertado en la selección de los genes que más influyen en cada caso.

Podemos concluir, por tanto, que la utilización en la clasificación final de una secuencia genética con un número de variables suficientemente grande (entre 100 y 300 variables) nos va a permitir extraer unos resultados realmente positivos. A partir de este conjunto inicial de genes, un incremento del tamaño de la secuencia no mejorará notablemente la tasa de error. Por el contrario, con la elección de una secuencia genética de menor longitud experimentaremos una importante degradación del rendimiento de ambos algoritmos.

CONCLUSION

Since the beginning, the main objective of our research was to develop an efficient algorithm able to extract the most relevant positions from a broad gene sequence. As we knew beforehand, it was going to be a hard task because we could only could work with a few patients whose DNA micro-arrays were composed of thousands elements. For this reason, we needed to find a classification and ranking method that could deal with a large number of features and a small number of training patterns.

Firstly we had to choose the classifier. Our aim was not only grouping the samples in their classes but also determining what criterion had been followed for that classification. In other words, we ought to find out what genes had resulted more conclusive in distinguishing the samples. It would help us to sort out the genes in the array to eliminate later those that were considered the most irrelevant to the diagnosis. We thought that a weight based linear classifier would be suitable to this problem so we decided to use the *state-of-art* algorithm, the *SVM*. Due to our problem could be studied as an easily separable case, we chose a linear *SVM* because of the nature of our variables.

After that, we proposed to use the *SVM* running together with the *recursive feature elimination* algorithm. The *RFE* would allow us to extract those less significant variables in the sequence and would establish an order of them. For this reason, we thought it would perform especially well due to those variables that did not provide too much information to this case would be eliminated. Once the process was finished we would have all the dimensions ranked in the sequence so we could classify the samples by using only those the algorithm considered as the most relevant.

Moreover we decided to implement a new version of the *RFE*, the robust *RFE*. It would be built by basing on the *bootstrap* technique and its main purpose would be achieving between a higher success rate in the classification process and an accurate ranking of the genes.

Despite of being slower than the standard *RFE*, we got a lot of improvements with our algorithm. On the one hand, we achieved a more accurate classification. Even though the *RFE* was quite efficient to this aim we were able to reduce the error rate so that hardly ever we made a mistake in the classification. Although our priority was not only having a low error rate, it was regarded as a very positive result. This was because of the data we were working with. Due to all databases were referred to cancer illnesses, it was of vital importance to avoid as much mistakes as possible. However, we have to mention again that our main objective was not to classify but to extract which were the most relevant genes and its dependences. We considered that achieving a high success rate was a way to check that we had been selecting the correct genes. We would only be able to test our success in the ranking of the genes by analyzing the results of the classification.

On the other hand we could ratify the actual relevance of each gene in the sequence. At the end of each algorithm we performed an average of the importance of the variables depending on the time when they were eliminated by the *RFE*. It let us determine that the genes were considered by the algorithm to be the most conclusive were always the same. Nevertheless, we faced the opposite situation with the unrelated variables. Due to the strong correlation among certain elements in the sequence, even the robust *RFE* was unable to select a concrete group as the less relevant. Because of this, we can observe great differences between the lower positions of the gene relevance average vectors for the two algorithms.

Finally, we had to remark that all these results had been compared for a process in which we only classified by using the 200 most conclusive variables. For that reason, we thought it would be useful to show how the algorithm worked with sequences of different sizes. By doing this, we could see that although all the genes provided a piece of information, choosing a large enough set of the most decisive would give us a very low error rate which would be perfectly suitable to our problem.

To sum up, we can conclude that although both methods are equally valid to resolve our problem, if we are looking for a more accurate classification and ranking of the genes, the robust *RFE* will always give us a better performance. Since the datasets we work with are all referred to serious diseases it is necessary to be as rigorous as possible with the results.

APÉNDICE A: PLANIFICACIÓN DEL PROYECTO

En este apartado vamos a presentar tanto la planificación como la estrategia de seguimiento empleada en este proyecto.

Comenzaremos planteando las etapas en las que se ha ido desarrollando el proyecto, desde su inicio hasta su conclusión. Para ello, vamos a incluir una tabla en la que describiremos cada una de las tareas que hemos llevado a cabo, basándonos en la fecha en la que comenzaron y la fecha en la que han concluido.

TAREA	FECHA INICIO	DURACIÓN (días)	FECHA FIN
FASE INICIAL			
Planteamiento del estudio y fijación de objetivos	20/10/2014	7	27/10/2014
Transferencia de conocimientos	28/10/2014	21	18/11/2014
Análisis de estudios previos	12/11/2014	14	26/11/2014
CLASIFICADORES			
Análisis de clasificadores	27/11/2014	28	25/12/2014
Estudio de las SVM	26/12/2014	14	09/01/2015
SVM lineal binaria	10/01/2015	14	24/01/2015
SVM lineal multiclase	25/01/2015	35	01/03/2015
DESARROLLO DE ALGORITMOS DE SELECCIÓN			
RFE	02/03/2015	42	13/04/2015
RFE robusto	14/04/2015	14	28/04/2015
EXPERIMENTOS			
Análisis de resultados para el primer conjunto de variables	29/04/2015	14	13/05/2015
Variación de parámetros	14/05/2015	7	21/05/2015
Comparación de resultados	22/05/2015	7	29/05/2015
MEMORIA	30/05/2015	20	19/06/2015

Tabla 7. Planificación del proyecto

A continuación hemos decidido introducir también un diagrama de Gantt con el objetivo de mostrar el tiempo de dedicación invertido en las diferentes tareas.

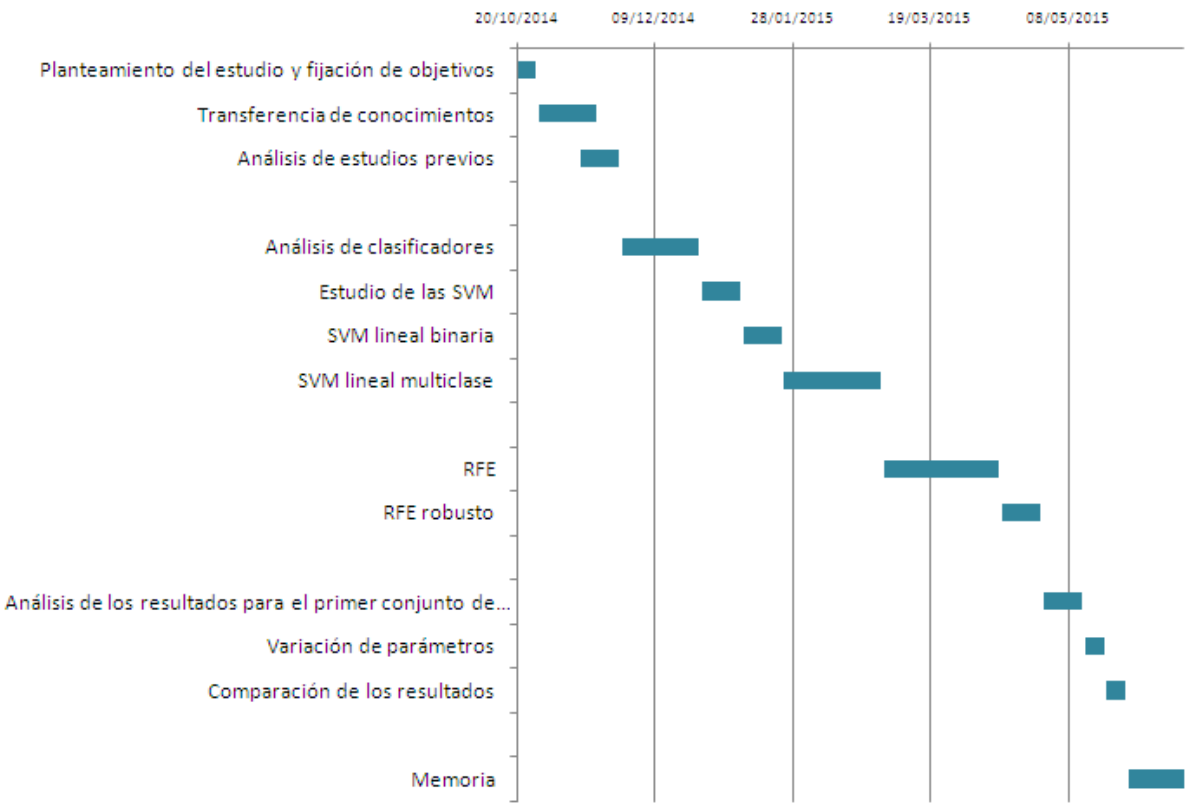


Tabla 8. Diagrama de Gantt

APÉNDICE B: PRESUPUESTO DEL PROYECTO

En el presente anexo vamos a exponer de manera detallada el desglose presupuestario que ha supuesto la elaboración de este proyecto, para lo cual será necesario distinguir entre los costes de personal y los costes de material.

COSTES DE PERSONAL

Con el objetivo de calcular de una manera más eficiente el gasto que supondría nuestro proyecto desde el punto de vista de los recursos humanos, vamos a incluir el cómputo de horas que invertimos en cada fase de realización del mismo.

FASE	NÚMERO DE HORAS
FASE INICIAL	196
CLASIFICADORES	476
ALGORITMOS DE SELECCIÓN	224
EXPERIMENTOS	112
MEMORIA	80

Tabla 9. Horas empleadas por cada fase del proyecto

Teniendo en cuenta que en la participación del proyecto han colaborado dos personas, un ingeniero junior y un tutor con rango de investigador, el coste correspondiente al personal corresponderá al valor mostrado en la siguiente tabla.

PERSONAL	HORAS DE TRABAJO	€/HORA	TOTAL
INGENIERO JUNIOR	938	11	10.318
TUTOR	150	20	3.000
TOTAL	**13.318 €**		

Tabla 10. Costes de personal

COSTES DE MATERIAL

Además de los costes descritos anteriormente, debemos considerar aquellos relativos al material del que nos hemos servido para el desarrollo de nuestro proyecto. Entre ellos

figuran, el equipo informático, la licencia del programa utilizado y la documentación relativa a estudios previos y a bases de datos a partir de las cuales nos basamos para llevar a cabo el estudio. Por otra parte, hemos de tener en cuenta también los gastos correspondientes a la tarifa de la luz y la conexión a Internet. En la siguiente tabla aparecen detallados.

CONCEPTO	PRECIO
Ordenador	500
Licencia Matlab (5 meses)	2300
Documentación	200
Luz (Iberdrola)	70
Conexión a Internet (Movistar)	98
TOTAL	**3.798 €**

Tabla 11. Costes de material

RESUMEN DE COSTES

Basándonos en todos los costes acumulados vamos a concluir con una tabla resumen con el presupuesto total estimado para la ejecución completa del proyecto.

CONCEPTO	PRECIO
COSTES DE PERSONAL	13.318
COSTES DE MATERIAL	3.798
IMPORTE BRUTO	17.116
IVA (21 %)	3.594,36
TOTAL	**20.710,36 €**

Tabla 12. Resumen de costes

APÉNDICE C: LIBSVM

Para elaborar nuestro estudio hemos decidido basarnos en una de las librerías de la *Statistics Toolbox* de Matlab, la *libsvm* [5]. Esta librería, que tiene su origen en el año 2000, fue desarrollada por la Universidad de Taiwan con el objetivo de resolver de manera más eficiente problemas de aprendizaje automático mediante máquinas de vectores de soporte.

La *libsvm* soporta varias formulaciones de SVM tanto para clasificación como para regresión o estimación de distribuciones. Para nuestro problema, nos centraremos únicamente en la clasificación (*C*-support vector classification, *C*-SVC), que se basa en dos etapas fundamentales:

1. Entrenar los datos de entrada para obtener un modelo.

2. Utilizar ese modelo para predecir los resultados de un conjunto de datos de test.

Vamos a explicar a continuación las dos funciones de esta librería que hemos utilizado para el desarrollo del proyecto.

- **svmtrain**

A esta función le pasaremos como datos de entrada un vector con etiquetas de entrenamiento, una matriz de kernels con las muestras de entrenamiento y un parámetro de opciones en el que incluiremos el tipo de clasificación (*C*-SVC), la utilización de una matriz de kernels y el valor de coste óptimo que habíamos calculado. Como resultado final obtendremos una estructura que incluirá, entre otros datos, el número de vectores de soporte, los índices de la matriz de muestras que representan vectores de soporte y los coeficientes α necesarios para el cálculo de los pesos.

- **svmpredict**

En el caso de esta función, tendremos que introducir el vector con la etiqueta de test, la matriz con los datos de test y la estructura que nos devolvió la *svmtrain*. Esto nos permitirá después quedarnos con la etiqueta predicha por el clasificador y la precisión del resultado.

BIBLIOGRAFÍA

[1] Guyon, I., Weston, J., Barnhill, S., (2002). Gene Selection for Cancer Classification Using Support Vector Machines. Barnhill Bioinformatics, Savannah, Georgia, USA.

[2] Guzmán, F., Ortega, M., (2006). Desarrollo y Evaluación de Métodos Constructivos Basados en el Discriminante Lineal de Fisher. Tesis. Departamento de Teoría de la Señal y Comunicaciones. Universidad Carlos III de Madrid.

[3] Brown, M., Noble, W., Lin, D., Cristianini, N., Sugnet, C., Ares, M., Haussler, D., (1999). Support Vector Machine Classification of Microarray Gene Expression Data. Department of Computer Science, University of California, Santa Cruz.

[4] Carrasco, A., Aler, R., (2009). Link Networks con un Enfoque Multiobjetivo. Tesis. Departamento de Teoría de la Señal y Comunicaciones. Universidad Carlos III de Madrid.

[5] Chang, C., Lin, C., (2014). LIBSVM: a simple and easy-to-use support vector machines tool for classification (C-SVC, nu-SVC), regression (epsilon-SVR, nu-SVR), and distribution estimation. Department of Computer Science, National Taiwan University.

[6] Nutt, C., Mani, D., Betensky, R., Tamayo, P., Cairncross, J., Ladd, C., Pohl, U., Hartmann, C., McLaughlin, M., Batchelor, T., Black, P., von Deimling, A., Pomeroy, S., Golub, T., Louis, D., (2003). Gene Expression based Classification of Malignant Gliomas Correlates Better with Survival than Histological Classification. Cancer Research, 63:1602-1607.

[7] Bhattacharjee, A., Richards, W., Staunton, J., Li, C., Monti, S., Vasa, P., Ladd, C., Beheshti, J., Bueno, R., Gillette, M., Loda, M., Weber, G., Mark, E., Lander, E., Wong, W., Johnson, B., Golub, T., Sugarbaker, D., Meyerson, M., (2001). Classification of human lung carcinomas by mRNA expression profiling reveals distinct adenocarcinoma subclasses. Proceedings of the National Academy of Sciences of USA, 98:13790-13795.

[8] Su, A., Welsh, J., Sapinoso, L., Kern, S., Dimitrov, P., Lapp, H., Schultz, P., Powell, S., Moskaluk, C., Frierson, H., Hampton, G., (2001). Molecular Classification of Human Carcinomas by Use of Gene Expression Signatures. Cancer Research, 61:7388-7393.

[9] Singh, K., Xie, M., (2004). Bootstrap: A Statistical Method. Rutgers University.

[10] Bishop, C., (2006). Pattern Recognition and Machine Learning. Springer.